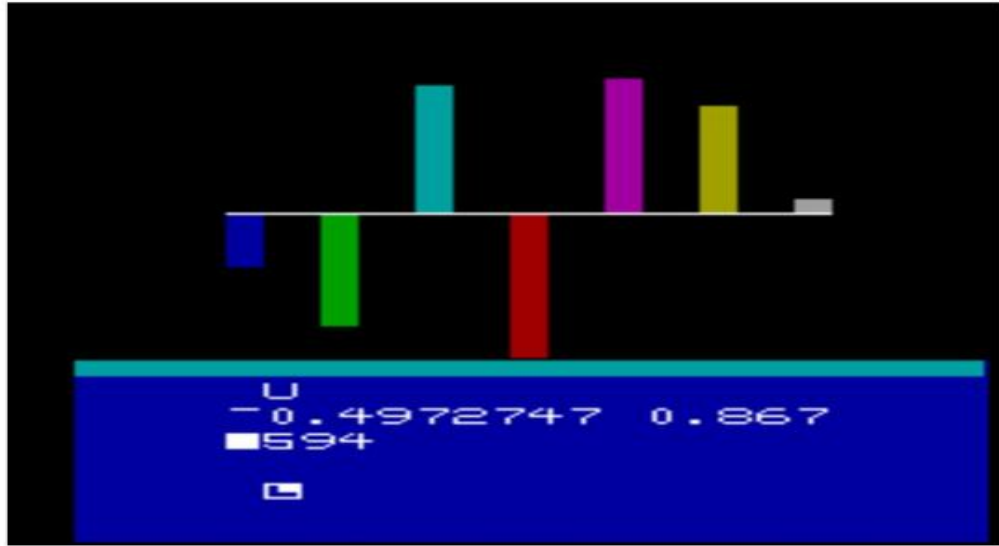




21 APL/S Games for Your Umtech VideoBrain

by James Price

21 APL/S Games and One-Liners for Your Umtech VideoBrain

APL Games for Microcomputers, Volume 1

James Price

Published by James Price, 2025.

While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

21 APL/S GAMES AND ONE-LINERS FOR YOUR UMTECH
VIDEOBRAIN

First edition. December 19, 2025.

Copyright © 2025 James Price.

Written by James Price.

Table of Contents

[Title Page](#)

[Copyright Page](#)

[21 APL/S Games and One-Liners for Your Umtech VideoBrain \(APL Games for Microcomputers, #1\)](#)

[Acknowledgments](#)

[Dedication](#)

[Series Introduction](#)

[Introduction to APL/S on the Umtech VideoBrain running on MAME](#)

[Slot Machine](#)

[Sine Barchart One-Liner](#)

[Acey Ducey](#)

[Roulette](#)

[Gunner](#)

[Collatz Bounce](#)

[Phi and Fibonacci One-Liners](#)

[Pyramid chart](#)

[Recursive Factorial Fun](#)

[Lunar Lander](#)

[Unit Circle](#)

[Dicejack](#)

[Primes](#)

[Data Statistics](#)

[Lotto Picker](#)

[Estimate Pi](#)

[Taylor, Maclaurin, and Madhava-Leibniz Series](#)

[Stag Hunt](#)

[Moo](#)

[Ecte's Guess](#)

[Recursive Coin Change Maker](#)

[Calendar Clock](#)

[Credits](#)

[The APL/S games in this book are collected here:](#)

[Sign up for James Price's Mailing List](#)

Copyright © 2025 James Price. All rights reserved.

This publication is an independent, unofficial work and is not authorized, licensed, sponsored, or endorsed by Umtech, Inc., or by any successor, assignee, or holder of intellectual property rights associated with the Umtech VideoBrain computer system. Umtech, Inc. is no longer in operation.

All programs contained herein are original works by the author unless otherwise indicated. Programs are written in APL/S for compatibility with the Umtech VideoBrain. No original Umtech software, firmware, ROM code, source code, or proprietary documentation is reproduced, distributed, or derived. Any trademarks or product names referenced are used solely for identification and compatibility purposes and remain the property of their respective owners. While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Acknowledgments

Thanks to Sean Riddle for making this all possible by performing the heavy lifting and technical magic needed to resurrect the APL/S ROMs so we can emulate The Computational Language of the Umtech VideoBrain using MAME. Thanks also to the APL/S cartridge donor, those on the MAME team who added the VideoBrain computer system to MAME, and the VideoBrain enthusiast community for keeping this system in their hearts all these years.

Thanks to The Array Cast team of Bob Therriault, Conor Hoekstra, Marshall Lochbaum, Adám Brudzewsky, and Stephen Taylor for having me on the show and encouraging me to push forward with writing APL games books. Thanks also to Marshall Lochbaum for his enthusiasm in documenting APL/S on the APL Wiki, and for hosting the Discord APL Farm “Quad Squad,” a weekly discussion of ACM SIGAPL Quote Quad articles, which lead to the port of the Unit Circle program.

Thanks to Elias Mårtenson for his encouragement and interest in this project and suggestions for improvements to the upcoming print edition.

Dedication

To my beloved tabby cat, Mr. Tibbs.

Thanks for hanging out with me while I was writing these games during late Summer and Autumn of 2025, little buddy.

Series Introduction

An early APL gaming culture left behind a rich treasure trove of artifacts now readily accessible to code archaeologists. Early issues of *Quote Quad* contained APL implementations of [Poker](#), [Target](#), [Conway's Game of Life](#), and [Tic Tac Toe](#). A game called [Pico-Fumi](#) is found in the *Proceedings of the Fifth International APL Users' Conference* (May 1973). Gaming historians digging through program listings on [MTS](#) systems running on IBM System/360s will uncover a [resource management game](#) called [Kingdom](#) written in APL. An issue of the I. P. Sharp newsletter called for submissions of [WARI](#) (Mancala) playing programs. The submitted entries then fought a series of battles, and the winners were reported in a [follow-up issue](#).

I.P. Sharp newsletters from the late 1970's document a [91 FUNSPACE](#) workspace, and track the evolution of 92 GAMES, which included [ELIZA](#), [STARTREK](#), [SPACEWAR](#), [GAMBLE](#), [SNIM](#), [WARGAMES](#), [WORDGAMES](#), [MASTERMIND](#), [QUADGT](#), [WUMPUSHUNT](#), and a game called “[Bridge Hands](#)”.

The I.P. Sharp [Public Workspaces Catalogue](#) entry for 92 GAMES from 1981 lists “tictactoe, blackjack, showdown poker, slot machine, craps, horserace, supernim, gunfire games, hangman, scramble, and finite-state machine, nim, mastermind, wumpus, and quadgit.” It also includes 93 LIFE: “[This workspace contains Life, a solitaire game involving a world in which cells live and die according to a few simple rules. The rules have been chosen to make the behaviour of the cell population quite interesting.](#)”

The MCM/70's APL library [LIB/70](#) included a “[Fun and Games Library](#)” in 1974 consisting of “[several games, such as HOR \(HORSE, possibly the world's earliest game implemented on a microcomputer\), HAN \(HANGMAN\), GUN, etc.](#)” Reviewing the [tape](#), one also finds the source code to games like [Moo](#), [Craps](#), and [Blackjack](#).

Back then, it was common to [type in](#) programs from print media. A [Lunar Lander](#) game appears in the APL-themed August 1977 issue of *Byte* magazine, immediately following Ken Iverson's article “Understanding

APL.” Type-in programs were published in magazines, like Creative Computing and [Compute’s Gazette](#), and in books of games written in BASIC by authors like [David Ahl](#) and [Tim Hartnell](#). BASIC was built into the microcomputers used in schools, Apple IIs and Commodore PETs, and in homes, Timex/Sinclair 1000s, Commodore 64s, and Atari 800XLs.

David Ahl explained in an [interview](#): “BASIC was the only widely available interactive language. Smalltalk? Lisp? Logo? MUMPS? They didn't stand a chance against BASIC.” Steve Wozniak recalls his motivation for writing Integer BASIC for the Apple computer: “There was a book out, 101 BASIC Computer Games, and Bill Gates had written in BASIC. So I said: [You've got to have these games; it's going to be the heart of a computer that's worth anything to people...](#)”

But with BASIC everywhere, many critics lamented the harm such a questionable language would inflict on vulnerable programming populations. Early use of BASIC was said to leave one “[mentally mutilated beyond hope of regeneration.](#)” Did [learning such a poorly crafted tool of thought first](#) leave [malleable young minds](#) permanently [misshapen](#)? To what extent did the use of BASIC represent a lost opportunity to introduce the microcomputer generation to structured programming?

What if, by a twist of fate, instead of BASIC, APL had been the language built into home microcomputers? What if an APL workspace had greeted programmers instead of the BASIC prompt? Like BASIC, APL is interpreted and well-suited to writing games. The Waterloo APL disk for the Commodore SuperPET shipped with a version of MASTERMIND, and the APL/Z80 disk included an implementation of Conway’s Game of Life.

So what might an APL games book have looked like, if APL had been the standard built-in language? These books explore APL in the spirit of learning the language by writing some of those games. This series will examine APL variants running on the Umtech VideoBrain, the Commodore SuperPET, the Commodore 64 and C128, and the MCM/70.

Introduction to APL/S on the Umtech VideoBrain running on MAME

The 1977 Umtech VideoBrain's programming language offering, The Computational Language cartridge, provided users with an interactive APL/S workspace with 920 bytes of free memory. All user programs and data, including variables and arrays, needed to fit into this tiny workspace. Numbers were stored as 4-byte floating points, with program keywords and variable references stored as single byte tokens.

Working within this very small workspace can be a challenge, especially when including visualization techniques provided by the primitive bar graph facility. Most of the games in this book use the bar graph feature in some way. Many of these games use almost every single byte of memory. Part of the fun in writing games for the VideoBrain is to see if you can squeeze in just one more feature, like adding leap year handling code to the clock calendar program. Sometimes a little space can be freed up by leaving initial variable assignments out of the program, and only typing them in before running it, or by shortening text strings used in output. The APL/S language is a good choice for code golfing.

Using MAME makes program development a lot easier, since you may save and load games using the debugger. The debugger can be used to examine memory and the stack. MAME even lets you speed up the system by skipping frames. You can remap keys, and I found the 'A' key needed to be remapped to the '.' key since the real 'A' key was inexplicably showing up as the 'O' key.

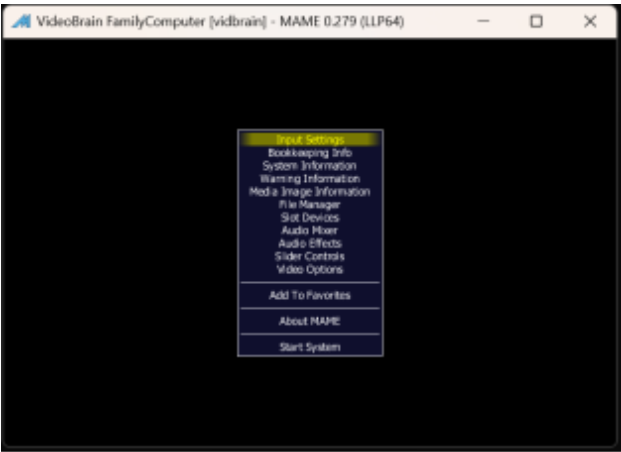
There are known problems with this system:

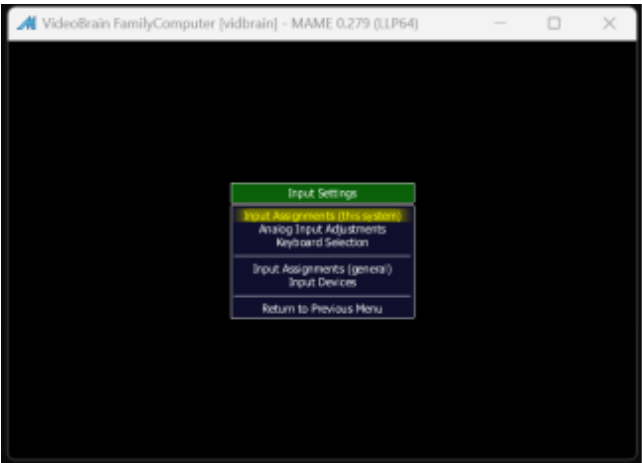
Imperfectly emulated features: sound

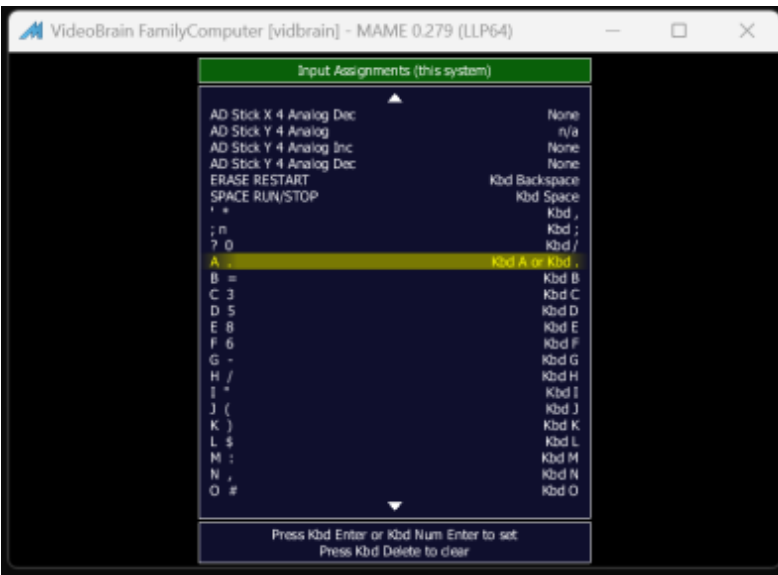
THIS SYSTEM DOESN'T WORK. The emulation for this system is not yet complete. There is nothing you can do to fix this problem except wait for the developers to improve the emulation.

Press any key to continue

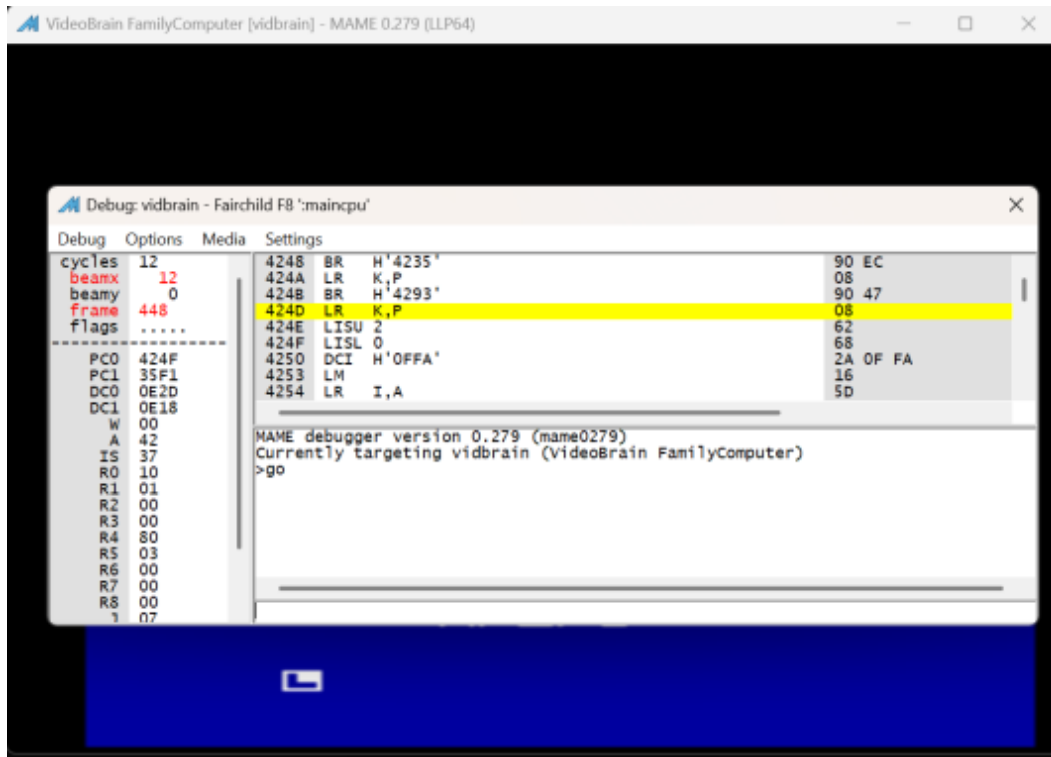
If you also need to remap the 'A' key, here's how: first, when loading The Computational Language cart, hit the TAB key here.







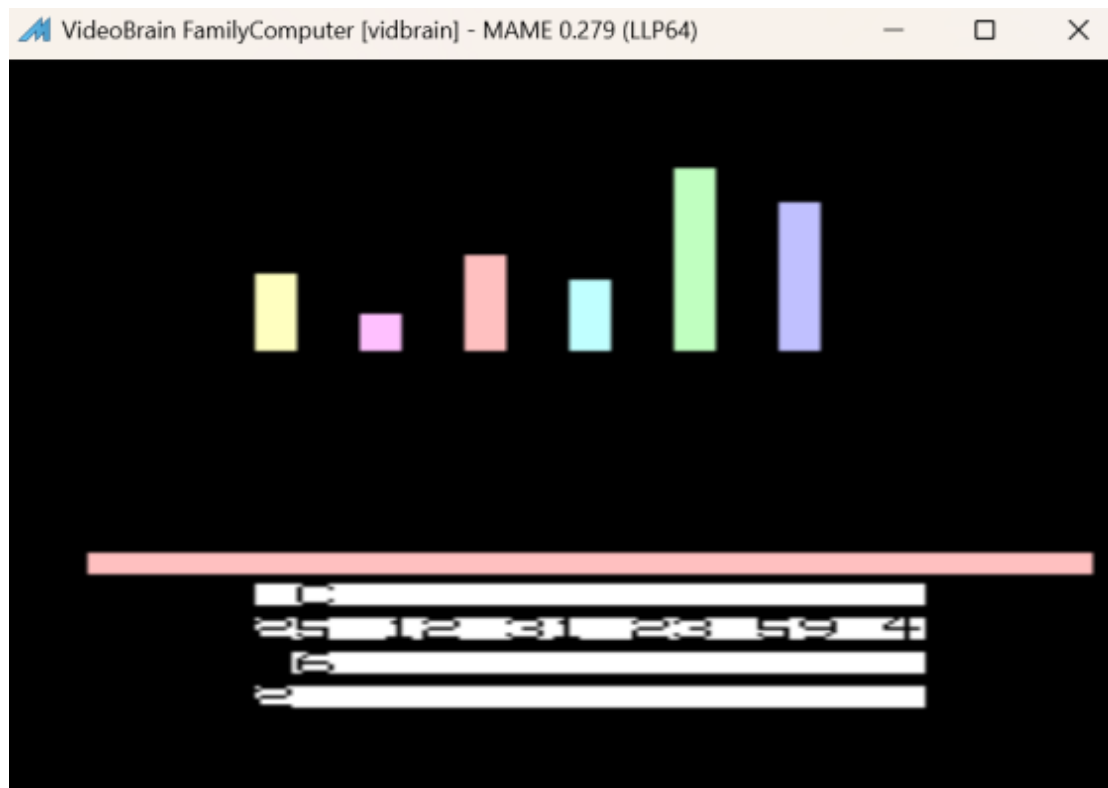
Then, go to the Input Settings for the keyboard keys, set the new ‘A’ key, and you’re good to go.



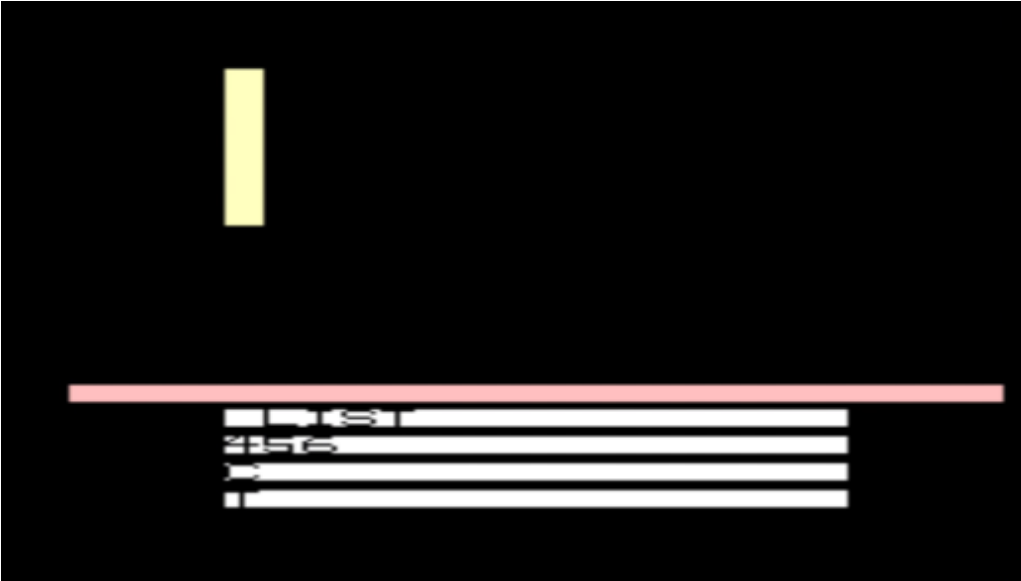
To load games in the debugger, launch MAME with the -debug option (you can also use the -window option to run MAME in a window). When the debugger starts up, type ‘go’ and hit return.



The game is now loaded, and ready to run. Hit the F3 key to execute the program.

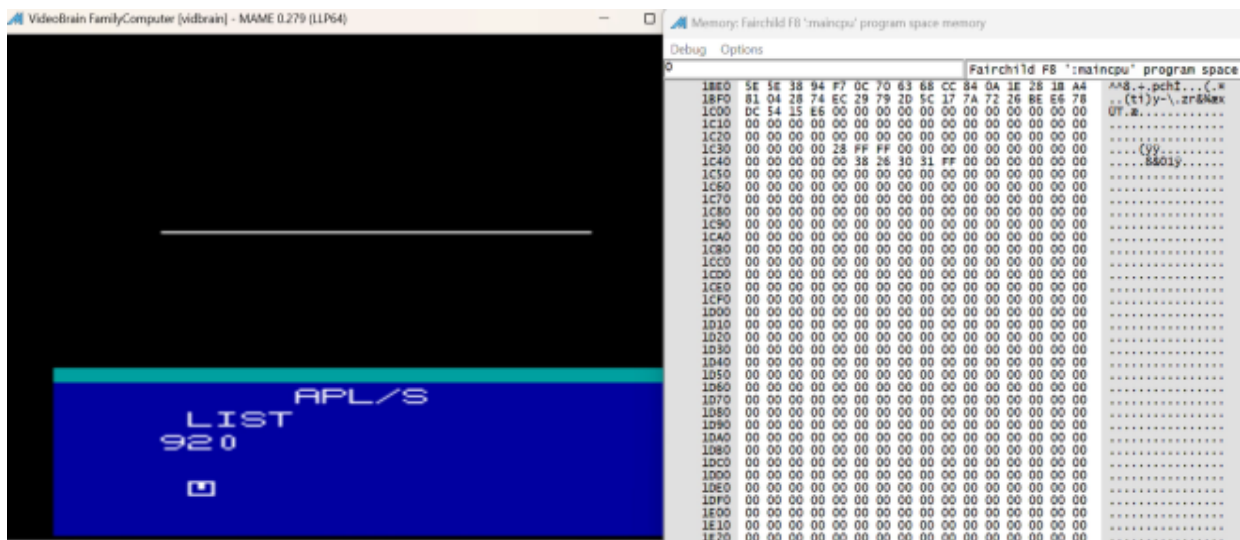


The program starts to run. The screen will look a little different than it usually looks after loading in a program using the debugger.

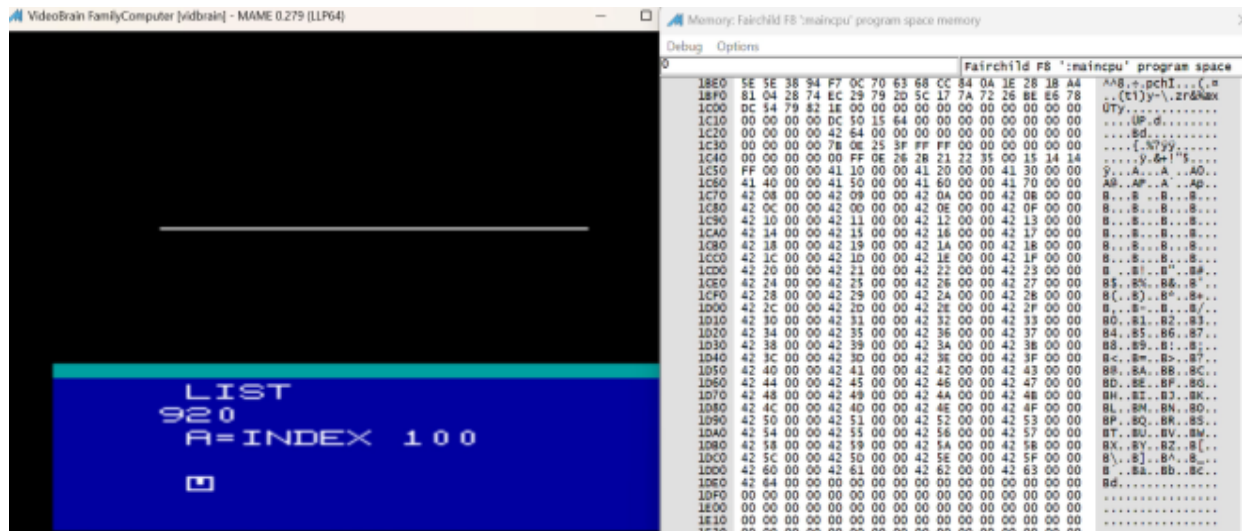


Alternatively, instead of running the program, you can LIST the variables and programs. Hit F1 to use the backspace to remove the program name, 'C', and then type LIST and hit F3 to enter the command.

LIST causes the free space remaining in the workspace to be shown, followed by the names of all programs and variables used.



The debugger can easily be used to examine programs and variables in memory. Click on Debug, then select New Memory Window to open a window to view the system memory.



We quickly fill up 400 bytes of the 920 bytes of free memory by creating an array of 100 numbers.

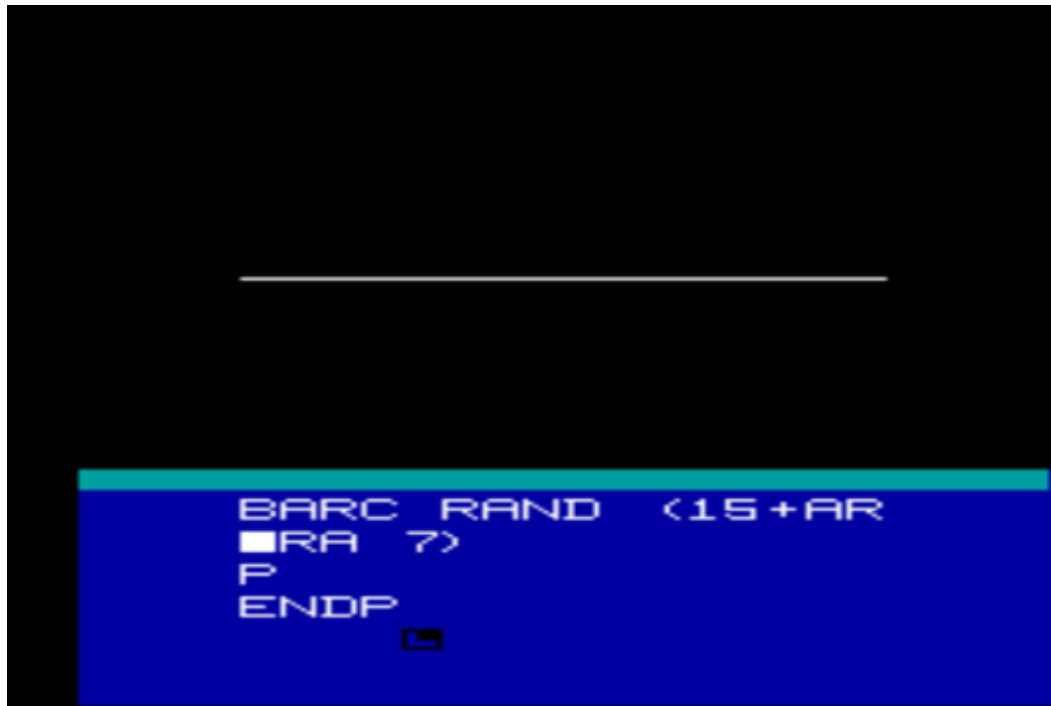
Watching the memory window during program runs can be very entertaining and offers insights into the operation of the system. We'll examine the stack operation in the Recursive Factorial Fun program chapter.

As far as I can tell, you cannot run the debug mode and use frame skipping at the same time. To speed up the system, run MAME without the debug flag, then increase system speed using frame skipping as usual. The only drawback seems to be that one cannot speed up a game loaded in using the debugger.

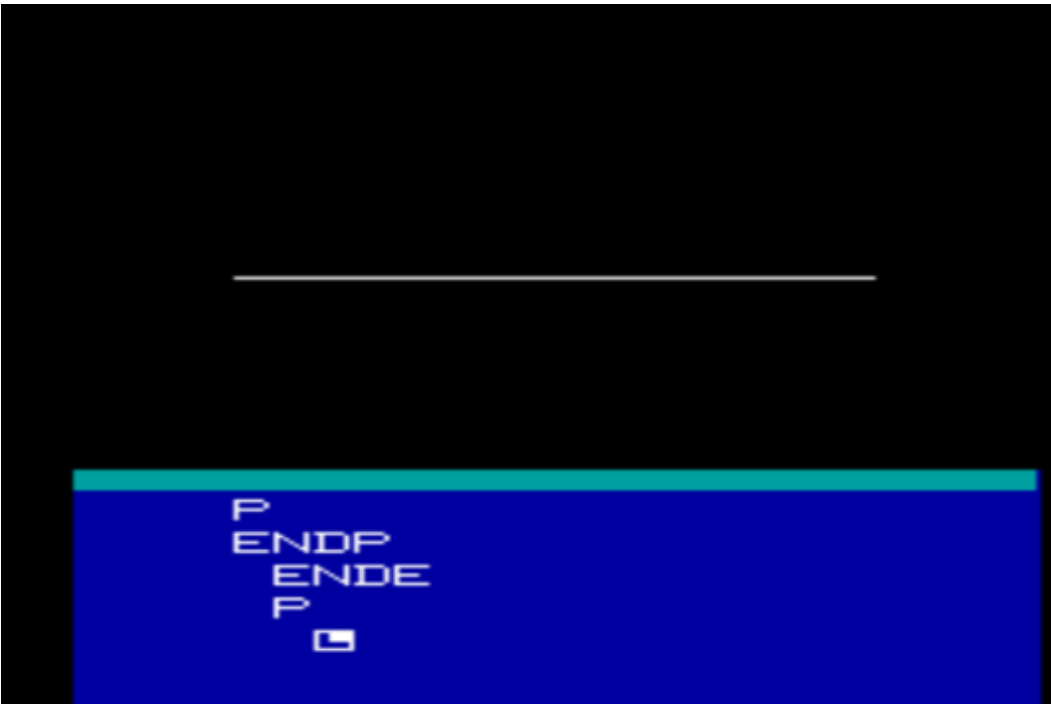
```
EDIT P
PROG P
  BARH RAND (64+A
  ■ RRA 7)
```

U

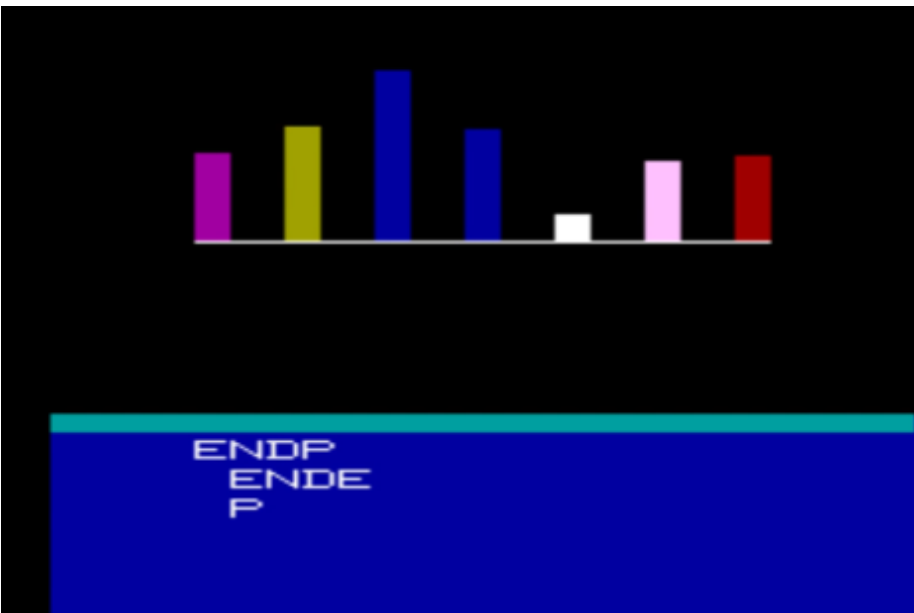
To enter in a new program, type EDIT PROGNAME, for example, EDIT P.



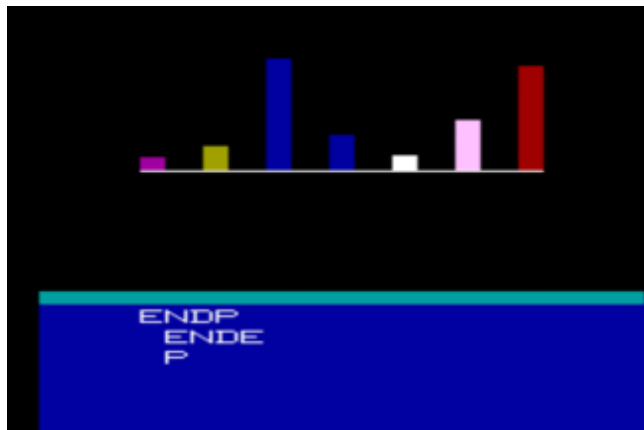
End the program with ENDP.

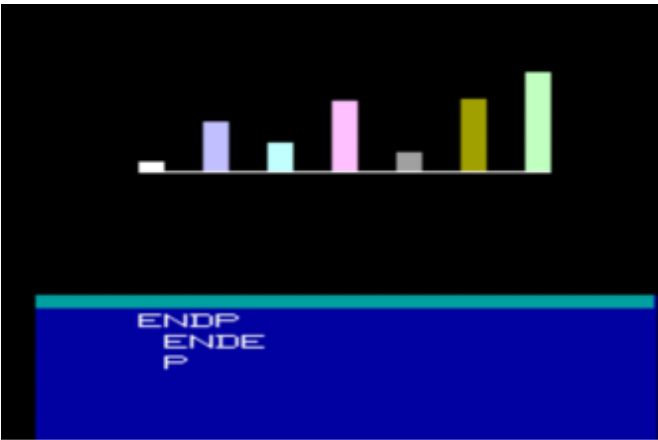


Type ENDE to exit the program editor. Then run the program by typing 'P' and hitting F3. (F2 moves the editor back one line, and F4 opens a new line in an existing program.)



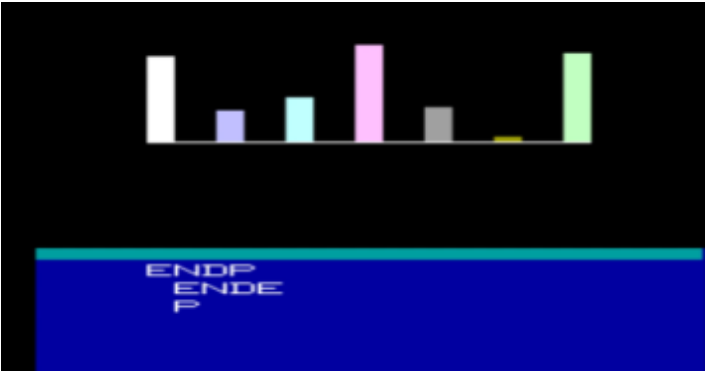
The program generates bar graphs of random height and color, then calls itself recursively.





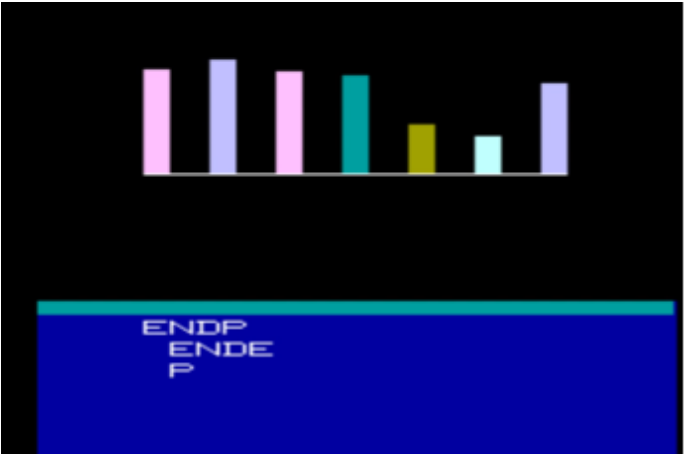


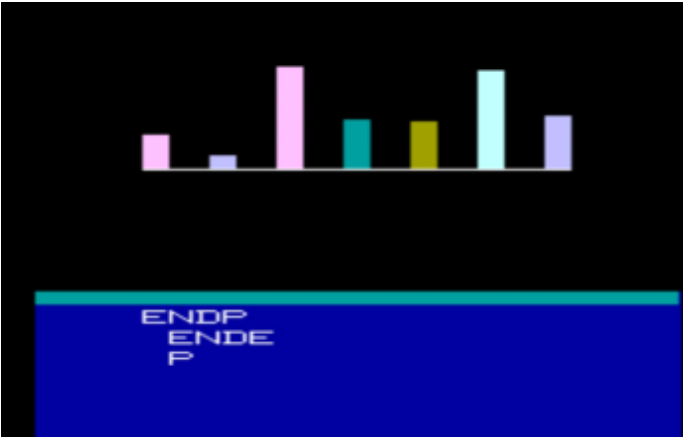


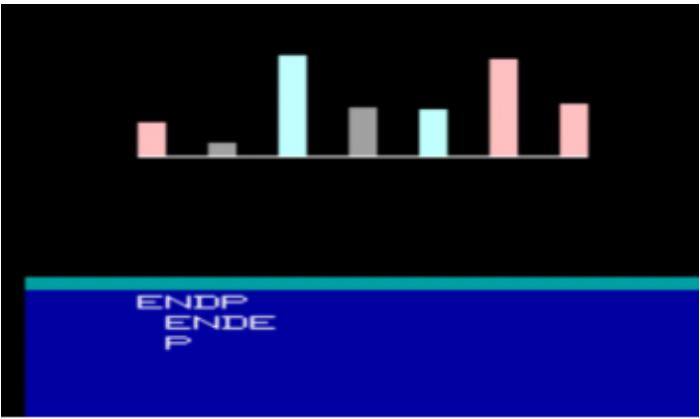


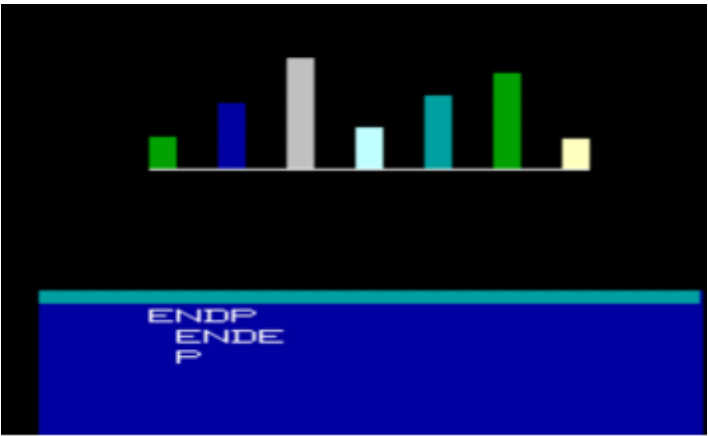












That should be enough information to get you started. I hope you have fun experimenting with the system, playing these games, and writing your own programs!

Slot Machine

A simplified port of the [David Ahl Basic Computer Games SLOTS](#) game.

```
PROG SLOT
CASH=100
BARH 20 20 20
WHILE 1
DO CASH
  "BET $1?"
  IF 0 GE BET=KEYB
  THEN EXIT
ENDIF
BARC RAND 6 6 6
BARC RAND 6 6 6
BARC RAND 6 6 6
BARC RAND 6 6 6
BARC RAND 6 6 6
BARC RAND 6 6 6
Z=RAND 6 6 6
BARC Z
I=+/ (Z(1) EQ Z)
IF I EQ 3
THEN
  "3 OF A KIND"
  CASH=CASH+5
ELSE
  IF I EQ 2
  THEN
    "2 OF A KIND"
    CASH=CASH+3
  ELSE
    J=(+/(Z(3) EQ Z))
    IF J EQ 2
    THEN
      "2 OF A KIND"
      CASH=CASH+3
    ENDIF
  ENDIF
ENDIF
CASH=CASH-1
ENDWHILE
ENDP
```

SLOTS

SLOT MACHINE

Description

The slot machine or one-arm bandit is a mechanical device that will absorb coins just about as fast as you can feed it. After inserting a coin, you pull a handle that sets three independent reels spinning. If the reels stop with certain symbols appearing in the pay line, you get a certain payoff. The original slot machine, called the Liberty Bell, was invented in 1895 by Charles Fey in San Francisco. Fey refused to sell or lease the manufacturing rights, so H.S. Mills in Chicago built a similar, but much improved, machine called the Operators Bell. This has survived nearly unchanged to today.

On the Operators Bell and other standard slot machines, there are 20 symbols on each wheel but they are not distributed evenly among the objects (cherries, bar, apples, etc.). Of the 8,000 possible combinations, the expected payoff (to the player) is 7,049 or \$69.11 for every \$100.00 put in, one of the lowest expected payoffs of all casino games.

In the program here, the payoff is considerably more liberal; indeed it favors the player by 11%--i.e., an expected payoff of \$11 for each \$100 bet. To approximate Nevada odds, reduce the jackpot to \$15 and keno to \$4.

SOURCE

Lots of slot machine programs were submitted including a very nice one by Bob Hoffberg of Roslyn, NY. The author of the one published is unknown.

PROGRAM LISTING

```
100 RANDOMIZE
110 DIM D(3)
120 PRINT "THIS IS A SIMULATION OF A SLOT MACHINE USING A COMPUTER "
130 PRINT "EACH TIME YOU 'PULL' I WILL ASK YOU IF YOU WISH TO PLAY AGAIN. "
140 PRINT "JUST ANSWER WITH A 'Y' FOR YES OR A 'N' FOR NO. "
150 PRINT "PLEASE PLACE 4 QUARTERS ON MY CPU FOR EACH PLAY. "
160 PRINT
170 FOR B1=1 TO 3
180 LET D(B1)=INT(RND(0)*6)+1
190 NEXT B1
200 FOR G1=1 TO 3
210 IF D(G1)=1 THEN 280
220 IF D(G1)=2 THEN 300
230 IF D(G1)=3 THEN 320
240 IF D(G1)=4 THEN 340
250 IF D(G1)=5 THEN 360
260 IF D(G1)=6 THEN 380
270 GOTO 580
280 PRINT TAB(G1*7); " BELL ";
290 GOTO 390
300 PRINT TAB(G1*7); " BAR ";
310 GOTO 390
320 PRINT TAB(G1*7); " CHERRY ";
330 GOTO 390
340 PRINT TAB(G1*7); " APPLE ";
350 GOTO 390
360 PRINT TAB(G1*7); " LEMON ";
370 GOTO 390
380 PRINT TAB(G1*7); " KENO ";
390 GOTO 390
400 PRINT
410 IF D(1)=1 AND D(2)=1 AND D(3)=1 THEN 420
420 PRINT "JACKPOT! 1000 COINS!"
430 GOTO 440
440 IF D(1)=1 AND D(2)=1 THEN 450
450 PRINT "TWO BELL! 100 COINS!"
460 GOTO 440
470 IF D(1)=1 AND D(2)=2 THEN 480
480 PRINT "BELL AND BAR! 50 COINS!"
490 GOTO 440
500 IF D(1)=1 AND D(2)=3 THEN 510
510 PRINT "BELL AND CHERRY! 25 COINS!"
520 GOTO 440
530 IF D(1)=1 AND D(2)=4 THEN 540
540 PRINT "BELL AND APPLE! 25 COINS!"
550 GOTO 440
560 IF D(1)=1 AND D(2)=5 THEN 570
570 PRINT "BELL AND LEMON! 25 COINS!"
580 GOTO 440
590 IF D(1)=1 AND D(2)=6 THEN 590
590 PRINT "BELL AND KENO! 25 COINS!"
600 GOTO 440
610 IF D(1)=2 AND D(2)=2 THEN 620
620 PRINT "TWO BAR! 50 COINS!"
630 GOTO 440
640 IF D(1)=2 AND D(2)=3 THEN 650
640 PRINT "BAR AND CHERRY! 25 COINS!"
650 GOTO 440
660 IF D(1)=2 AND D(2)=4 THEN 670
660 PRINT "BAR AND APPLE! 25 COINS!"
670 GOTO 440
680 IF D(1)=2 AND D(2)=5 THEN 690
680 PRINT "BAR AND LEMON! 25 COINS!"
690 GOTO 440
690 IF D(1)=2 AND D(2)=6 THEN 690
690 PRINT "BAR AND KENO! 25 COINS!"
700 GOTO 440
710 IF D(1)=3 AND D(2)=3 THEN 720
710 PRINT "TWO CHERRY! 50 COINS!"
720 GOTO 440
730 IF D(1)=3 AND D(2)=4 THEN 740
730 PRINT "CHERRY AND APPLE! 25 COINS!"
740 GOTO 440
750 IF D(1)=3 AND D(2)=5 THEN 760
750 PRINT "CHERRY AND LEMON! 25 COINS!"
760 GOTO 440
760 IF D(1)=3 AND D(2)=6 THEN 760
760 PRINT "CHERRY AND KENO! 25 COINS!"
770 GOTO 440
780 IF D(1)=4 AND D(2)=4 THEN 790
780 PRINT "TWO APPLE! 50 COINS!"
790 GOTO 440
790 IF D(1)=4 AND D(2)=5 THEN 800
790 PRINT "APPLE AND LEMON! 25 COINS!"
800 GOTO 440
800 IF D(1)=4 AND D(2)=6 THEN 800
800 PRINT "APPLE AND KENO! 25 COINS!"
810 GOTO 440
810 IF D(1)=5 AND D(2)=5 THEN 820
810 PRINT "TWO LEMON! 50 COINS!"
820 GOTO 440
820 IF D(1)=5 AND D(2)=6 THEN 830
820 PRINT "LEMON AND KENO! 25 COINS!"
830 GOTO 440
830 IF D(1)=6 AND D(2)=6 THEN 840
840 PRINT "TWO KENO! 50 COINS!"
840 GOTO 440
840 IF D(1)=6 AND D(2)=1 THEN 850
840 PRINT "KENO AND BELL! 25 COINS!"
850 GOTO 440
850 IF D(1)=6 AND D(2)=2 THEN 860
850 PRINT "KENO AND BAR! 25 COINS!"
860 GOTO 440
860 IF D(1)=6 AND D(2)=3 THEN 870
860 PRINT "KENO AND CHERRY! 25 COINS!"
870 GOTO 440
870 IF D(1)=6 AND D(2)=4 THEN 880
870 PRINT "KENO AND APPLE! 25 COINS!"
880 GOTO 440
880 IF D(1)=6 AND D(2)=5 THEN 890
880 PRINT "KENO AND LEMON! 25 COINS!"
890 GOTO 440
890 IF D(1)=6 AND D(2)=6 THEN 900
890 PRINT "KENO AND KENO! 25 COINS!"
900 GOTO 440
910 GOTO 170
920 PRINT "END OF PROGRAM"
930 GOTO 940
940 PRINT "THANK YOU FOR PLAYING"
950 GOTO 960
960 PRINT "GOOD BYE"
970 GOTO 980
980 PRINT "PROGRAM END"
```

An error in the original listing.

PROGRAM LISTING

```
100 RANDOMIZE
110 DIM D(3)
120 PRINT "THIS IS A SIMULATION OF A SLOT MACHINE USING A COMPUTER "
130 PRINT "EACH TIME YOU 'PULL' I WILL ASK YOU IF YOU WISH TO PLAY AGAIN. "
140 PRINT "JUST ANSWER WITH A 'Y' FOR YES OR A 'N' FOR NO. "
150 PRINT "PLEASE PLACE 4 QUARTERS ON MY CPU FOR EACH PLAY. "
160 PRINT
170 FOR B1=1 TO 3
180 LET D(B1)=INT(RND(0)*6)+1
190 NEXT B1
200 FOR G1=1 TO 3
210 IF D(G1)=1 THEN 280
220 IF D(G1)=2 THEN 300
230 IF D(G1)=3 THEN 320
240 IF D(G1)=4 THEN 340
250 IF D(G1)=5 THEN 360
260 IF D(G1)=6 THEN 380
270 GOTO 580
280 PRINT TAB(G1*7); " BELL ";
290 GOTO 390
300 PRINT TAB(G1*7); " BAR ";
310 GOTO 390
320 PRINT TAB(G1*7); " CHERRY ";
330 GOTO 390
340 PRINT TAB(G1*7); " APPLE ";
350 GOTO 390
360 PRINT TAB(G1*7); " LEMON ";
370 GOTO 390
380 PRINT TAB(G1*7); " KENO ";
390 GOTO 390
400 PRINT
410 IF D(1)=1 AND D(2)=1 AND D(3)=1 THEN 420
420 PRINT "JACKPOT! 1000 COINS!"
430 GOTO 440
440 IF D(1)=1 AND D(2)=1 THEN 450
450 PRINT "TWO BELL! 100 COINS!"
460 GOTO 440
470 IF D(1)=1 AND D(2)=2 THEN 480
480 PRINT "BELL AND BAR! 50 COINS!"
490 GOTO 440
500 IF D(1)=1 AND D(2)=3 THEN 510
510 PRINT "BELL AND CHERRY! 25 COINS!"
520 GOTO 440
530 IF D(1)=1 AND D(2)=4 THEN 540
540 PRINT "BELL AND APPLE! 25 COINS!"
550 GOTO 440
560 IF D(1)=1 AND D(2)=5 THEN 570
570 PRINT "BELL AND LEMON! 25 COINS!"
580 GOTO 440
590 IF D(1)=1 AND D(2)=6 THEN 590
590 PRINT "BELL AND KENO! 25 COINS!"
600 GOTO 440
610 IF D(1)=2 AND D(2)=2 THEN 620
620 PRINT "TWO BAR! 50 COINS!"
630 GOTO 440
640 IF D(1)=2 AND D(2)=3 THEN 650
640 PRINT "BAR AND CHERRY! 25 COINS!"
650 GOTO 440
660 IF D(1)=2 AND D(2)=4 THEN 670
660 PRINT "BAR AND APPLE! 25 COINS!"
670 GOTO 440
680 IF D(1)=2 AND D(2)=5 THEN 690
680 PRINT "BAR AND LEMON! 25 COINS!"
690 GOTO 440
690 IF D(1)=2 AND D(2)=6 THEN 690
690 PRINT "BAR AND KENO! 25 COINS!"
700 GOTO 440
710 IF D(1)=3 AND D(2)=3 THEN 720
710 PRINT "TWO CHERRY! 50 COINS!"
720 GOTO 440
730 IF D(1)=3 AND D(2)=4 THEN 740
730 PRINT "CHERRY AND APPLE! 25 COINS!"
740 GOTO 440
750 IF D(1)=3 AND D(2)=5 THEN 760
750 PRINT "CHERRY AND LEMON! 25 COINS!"
760 GOTO 440
760 IF D(1)=3 AND D(2)=6 THEN 760
760 PRINT "CHERRY AND KENO! 25 COINS!"
770 GOTO 440
780 IF D(1)=4 AND D(2)=4 THEN 790
780 PRINT "TWO APPLE! 50 COINS!"
790 GOTO 440
790 IF D(1)=4 AND D(2)=5 THEN 800
790 PRINT "APPLE AND LEMON! 25 COINS!"
800 GOTO 440
800 IF D(1)=4 AND D(2)=6 THEN 800
800 PRINT "APPLE AND KENO! 25 COINS!"
810 GOTO 440
810 IF D(1)=5 AND D(2)=5 THEN 820
810 PRINT "TWO LEMON! 50 COINS!"
820 GOTO 440
820 IF D(1)=5 AND D(2)=6 THEN 830
820 PRINT "LEMON AND KENO! 25 COINS!"
830 GOTO 440
830 IF D(1)=6 AND D(2)=6 THEN 840
840 PRINT "TWO KENO! 50 COINS!"
840 GOTO 440
840 IF D(1)=6 AND D(2)=1 THEN 850
840 PRINT "KENO AND BELL! 25 COINS!"
850 GOTO 440
850 IF D(1)=6 AND D(2)=2 THEN 860
850 PRINT "KENO AND BAR! 25 COINS!"
860 GOTO 440
860 IF D(1)=6 AND D(2)=3 THEN 870
860 PRINT "KENO AND CHERRY! 25 COINS!"
870 GOTO 440
870 IF D(1)=6 AND D(2)=4 THEN 880
870 PRINT "KENO AND APPLE! 25 COINS!"
880 GOTO 440
880 IF D(1)=6 AND D(2)=5 THEN 890
880 PRINT "KENO AND LEMON! 25 COINS!"
890 GOTO 440
890 IF D(1)=6 AND D(2)=6 THEN 900
890 PRINT "KENO AND KENO! 25 COINS!"
900 GOTO 440
910 GOTO 170
920 PRINT "END OF PROGRAM"
930 GOTO 940
940 PRINT "THANK YOU FOR PLAYING"
950 GOTO 960
960 PRINT "GOOD BYE"
970 GOTO 980
980 PRINT "PROGRAM END"
```

SLOT
100
BET \$1?
?1
□



?1
99
BET \$1?
?1
□



?1
98
BET \$1?
?1
□



?1
97
BET \$1?
?1
□



?1
2 OF A KIND
99
BET \$1?



?1
98
BET \$1?
?1
☐



?1
2 OF A KIND
100
BET \$1?



?1
99
BET \$1?
?1
□



?1
98
BET \$1?
?1
□



?1
3 OF A KIND
103
BET \$1?



?1
102
BET \$1?
?1
□



2 OF A KIND
104
BET \$1?
?1
□



?1
103
BET \$1?
?1
□



2 OF A KIND
105
BET \$1?
?1
□



?1
104
BET \$1?
?1
□



?1
103
BET \$1?
?1
□



2 OF A KIND
105
BET \$1?
?1
□



?1
104
BET \$1?
?1
☐



?1
2 OF A KIND
106
BET \$1?

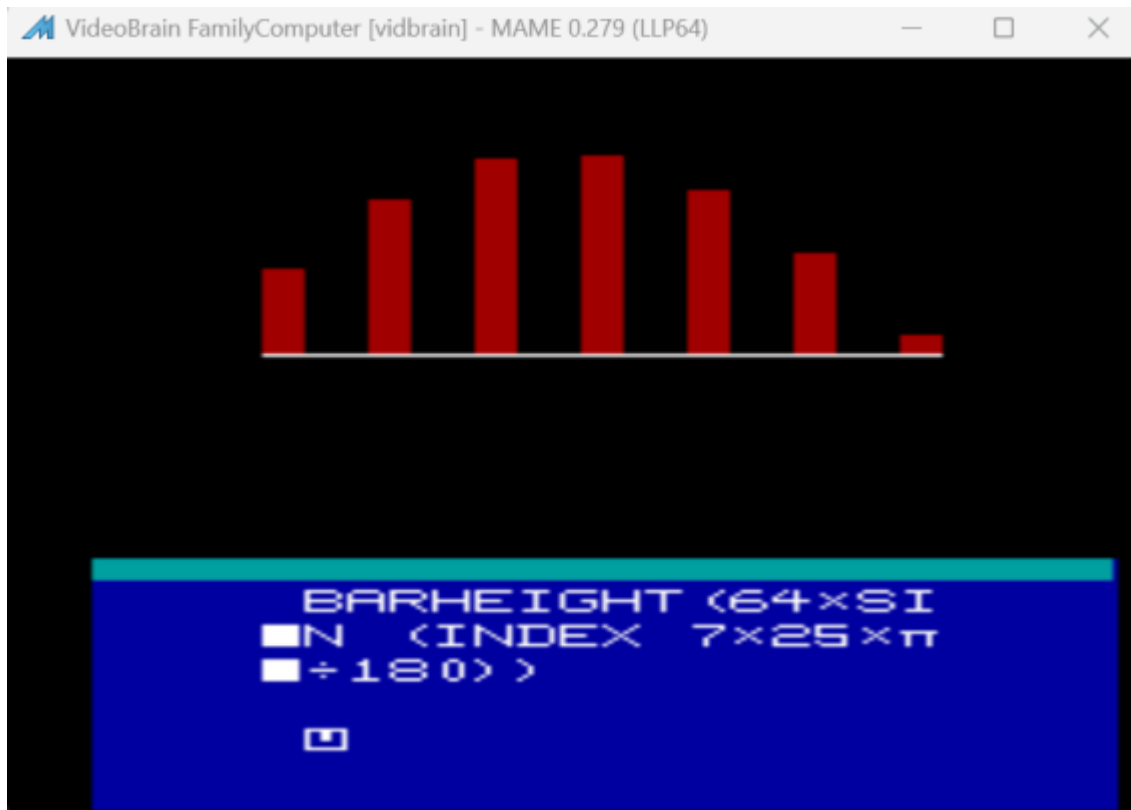


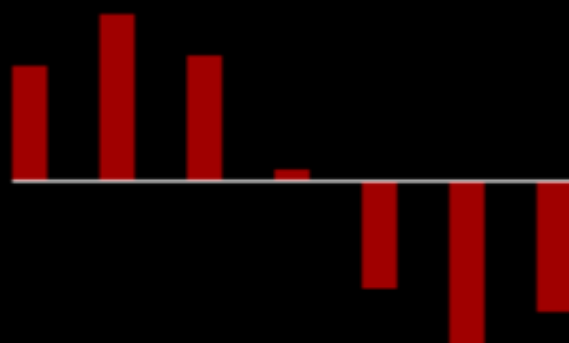
?1
105
BET \$1?
?1
☐



2 OF A KIND
107
BET \$1?
?
□

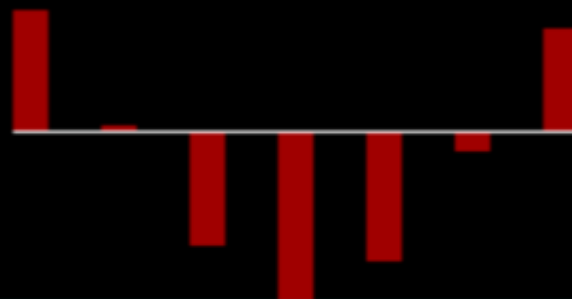
Sine Barchart One-Liner





APL/S
BARH 64×SIN (IND
■E 7×22×π+90)

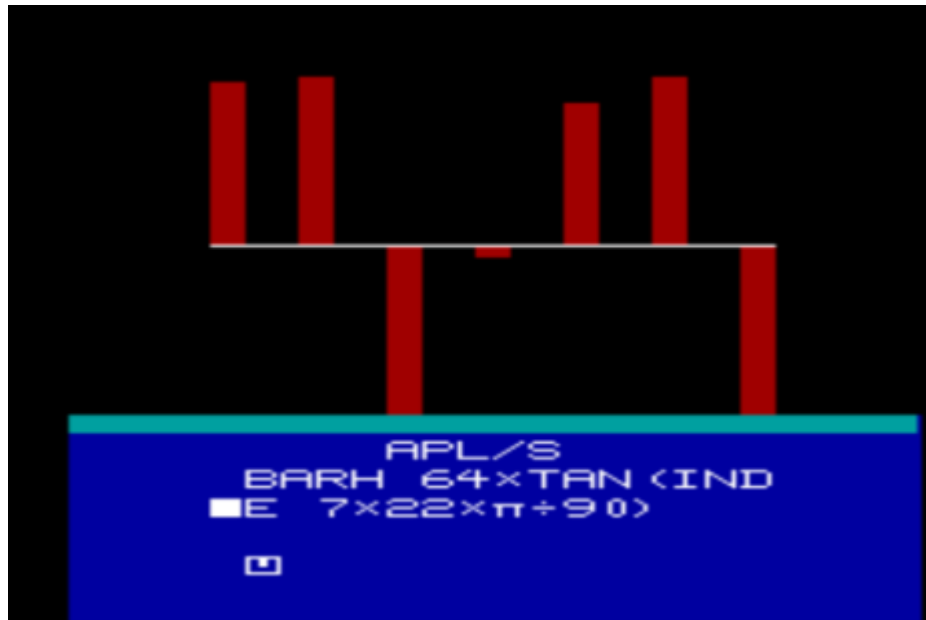




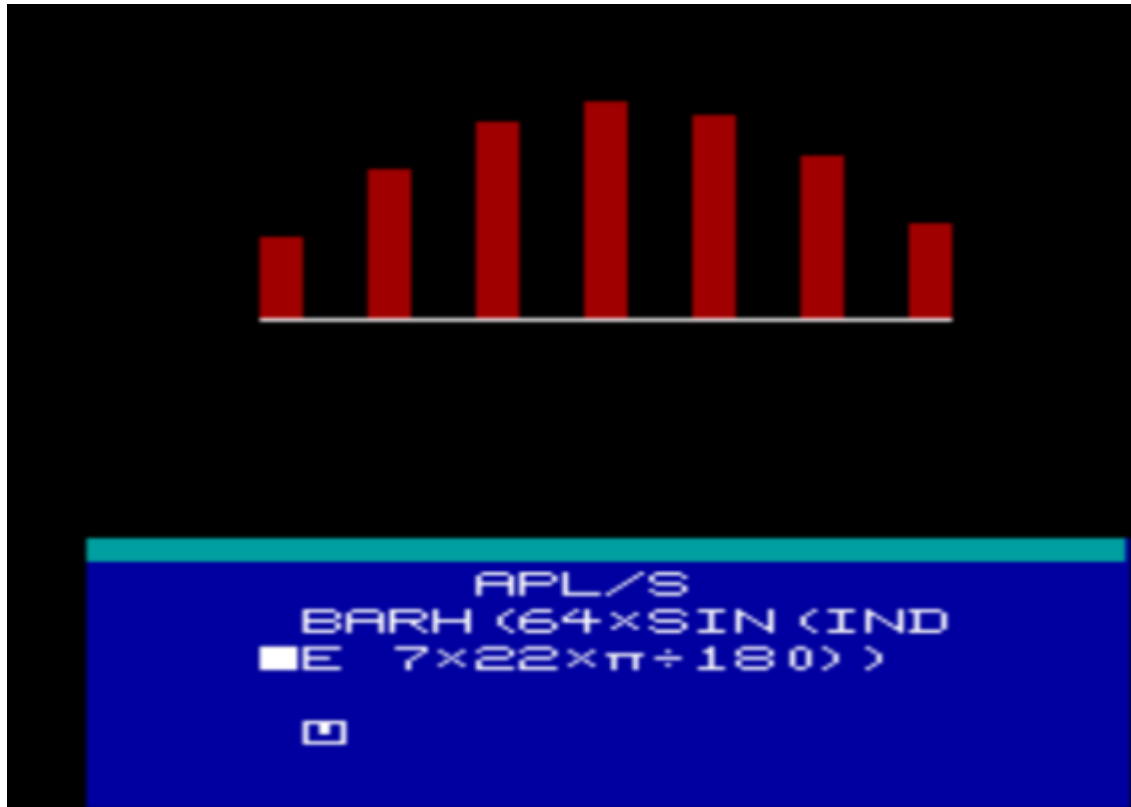
APL/S
BARH 64×COS (IND
■E 7×22×π÷90)



Cosine bar charted similarly.



Tangent graphed at the same points.



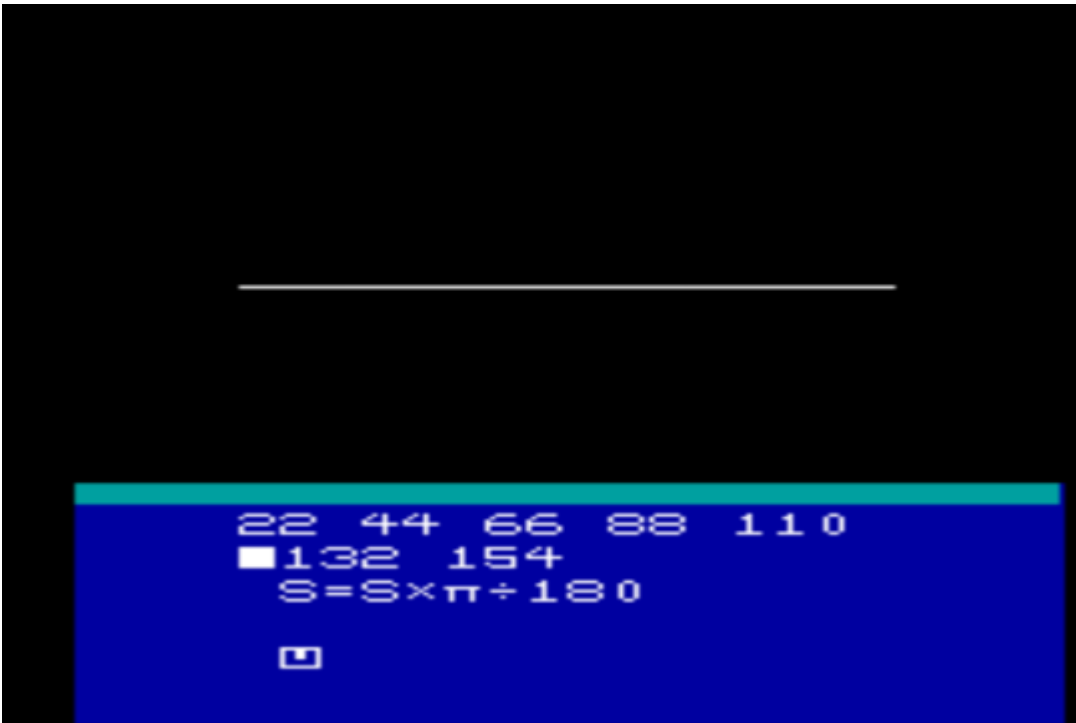
Change the sine chart's pi divisor from 90 to 180 to focus only on the positive value portion.



Let's walk through it, step by step.



Sample the sine curve using 7 values between 0 and 180 degrees.



Now convert from degrees to radians by multiplying by pi and dividing by 180.

■ .303834 2.68780

■ 8

S=SIN S

□



Take the
sine of each of the 7 values in the array.



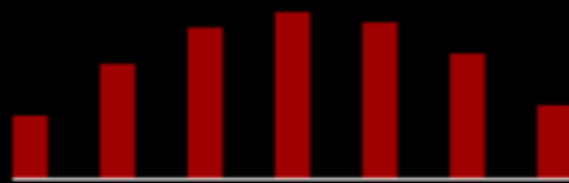
Scale up to 64 since the bar chart holds values between -64 and 64.



After scaling.

■3	58.4669	63.96
■1	60.1403	47.56
■134	28.05571	





■ 1 60.1403 47.56
■ 134 28.05571
BARH S



Now chart the scaled values. They do not need to be integers.



■ 134 28.05571
BARH S
BARC INDE 7
□

Change colors with a simple BARC using INDEX 7.

Acey Ducey

A port of David Ahl's 1973 [ACEYDU](#) from "101 BASIC Computer Games"

```

PROG ACEY
BARC 3 4 3
N=100
Q=100
WHILE Q GT 0
DO
  "YOUR CASH"
  Q
  "YOUR CARDS"
  A=RAND 13 13 13
  WHILE A(1) GE A(3)
  DO
    A=RAND 13 13 13
  ENDW
  Z=A(1)
  CARD
  Z=A(3)
  CARD
  BARH 4.5x(A(1),0,A(3))
  "BET?"
  M=KEYB
  IF M EQ 99
  THEN EXIT
  ENDIF
  Z=A(2)
  CARD
  IF (Z GT A(1)) AND (Z LT A(3))
  THEN "YOU WIN"
    Q=Q+M
    BARC 3 2 3
  ELSE "YOU LOSE"
    Q=Q-M
    BARC 3 4 3
  ENDIF
  BARH 4.5xA
ENDW
ENDP

```

The Acey Ducey main program.

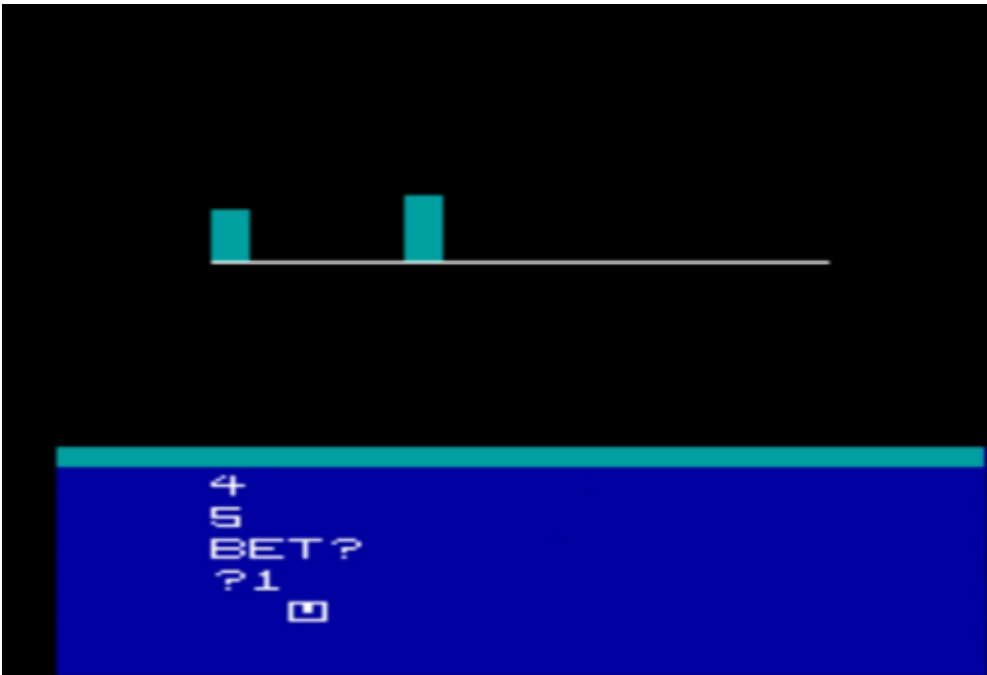
```
PROG CARD
IF (Z NE 1) AND (Z LT 11)
THEN Z
ELSE
  IF Z EQ 1
  THEN "ACE"
  ELSE
    IF Z EQ 11
    THEN "JACK"
    ELSE
      IF Z EQ 12
      THEN "QUEEN"
      ELSE "KING"
      ENDI
    ENDI
  ENDI
ENDI
ENDP
```


The main program calls the CARD subprogram to generate cards.



ACEY
YOUR CASH
100
YOUR CARDS

A sample game.







YOUR CASH
99
YOUR CARDS





2
YOU WIN
YOUR CASH



149
YOUR CARDS
2
QUEEN



2
QUEEN
BET?
?100

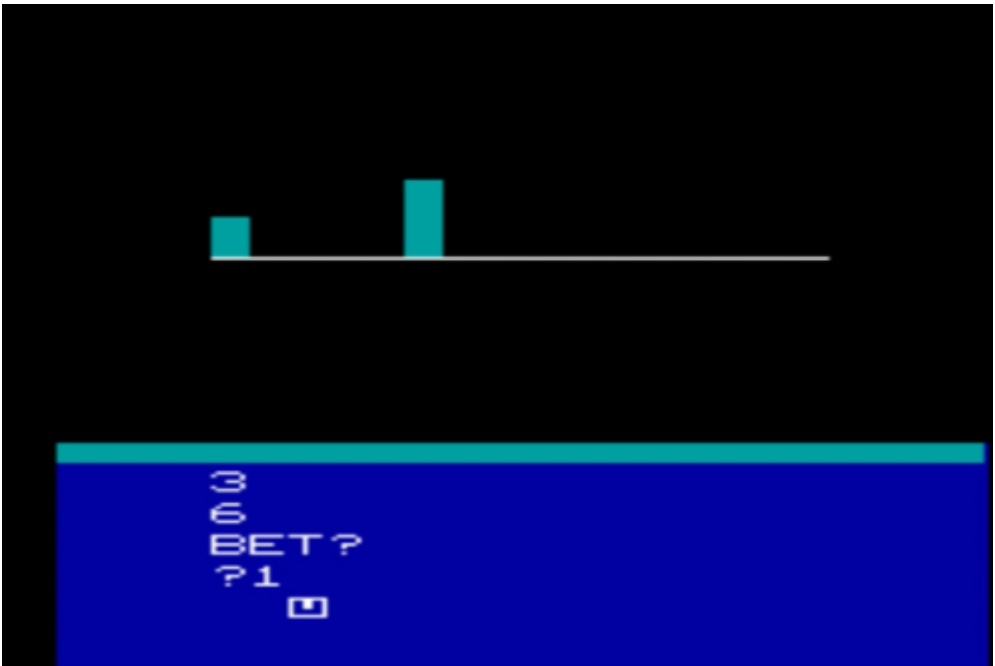




7
YOU WIN
YOUR CASH



YOUR CASH
249
YOUR CARDS

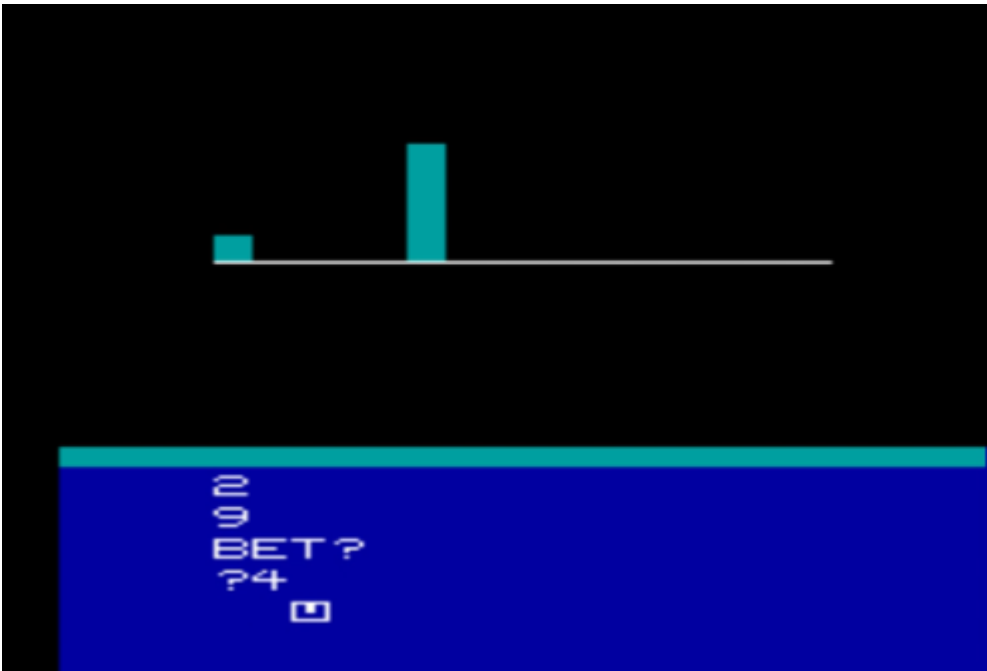


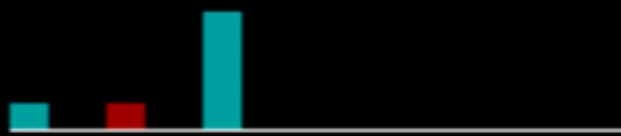


?1
5
YOU WIN



YOUR CASH
25 0
YOUR CARDS





2
YOU LOSE
YOUR CASH



246
YOUR CARDS
ACE
5



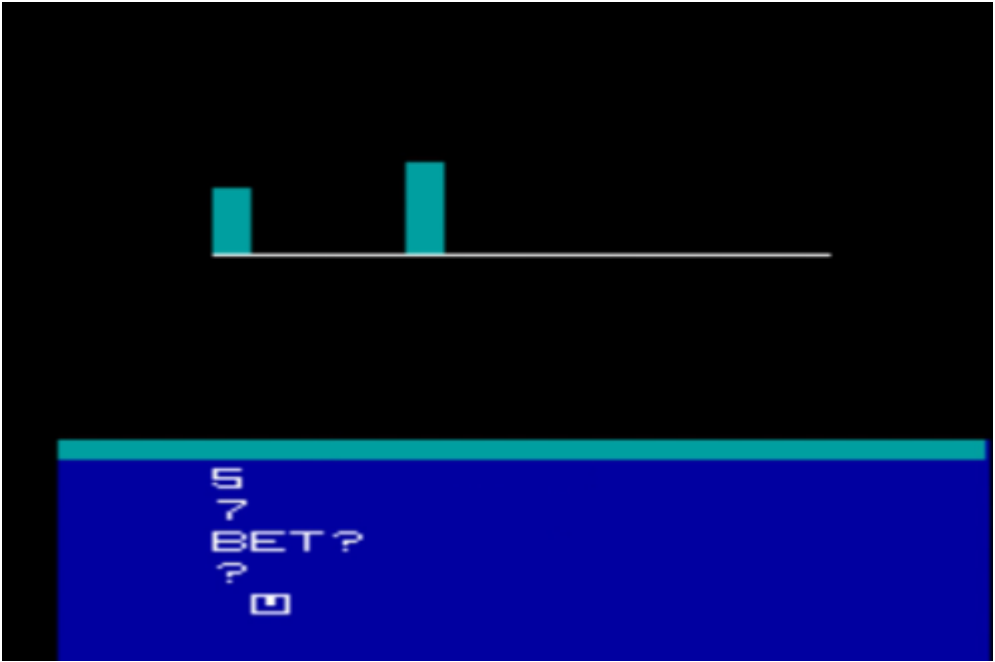
ACE
5
BET?
?5
□



3
YOU WIN
YOUR CASH



251
YOUR CARDS
5
7



ACEY
YOUR CASH
1 0 0

Another sample game.





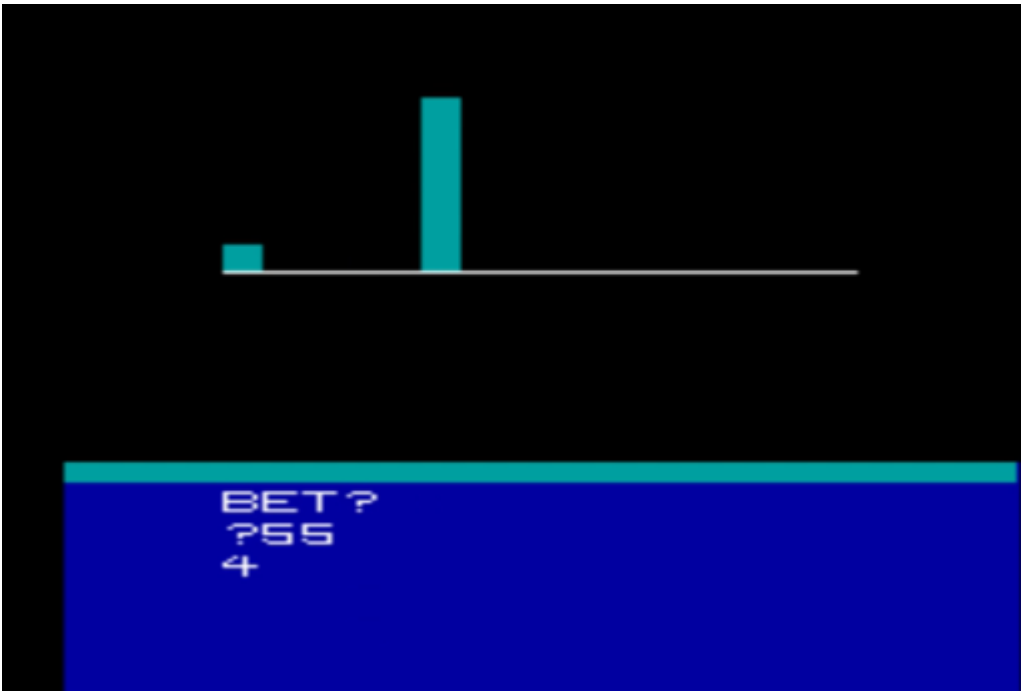
BET ?
?5
2



2
YOU LOSE
YOUR CASH
95



YOUR CARDS
2
KING





4
YOU WIN
YOUR CASH



150
YOUR CARDS
8





?S
QUEEN
YOU WIN



YOUR CASH
155
YOUR CARDS



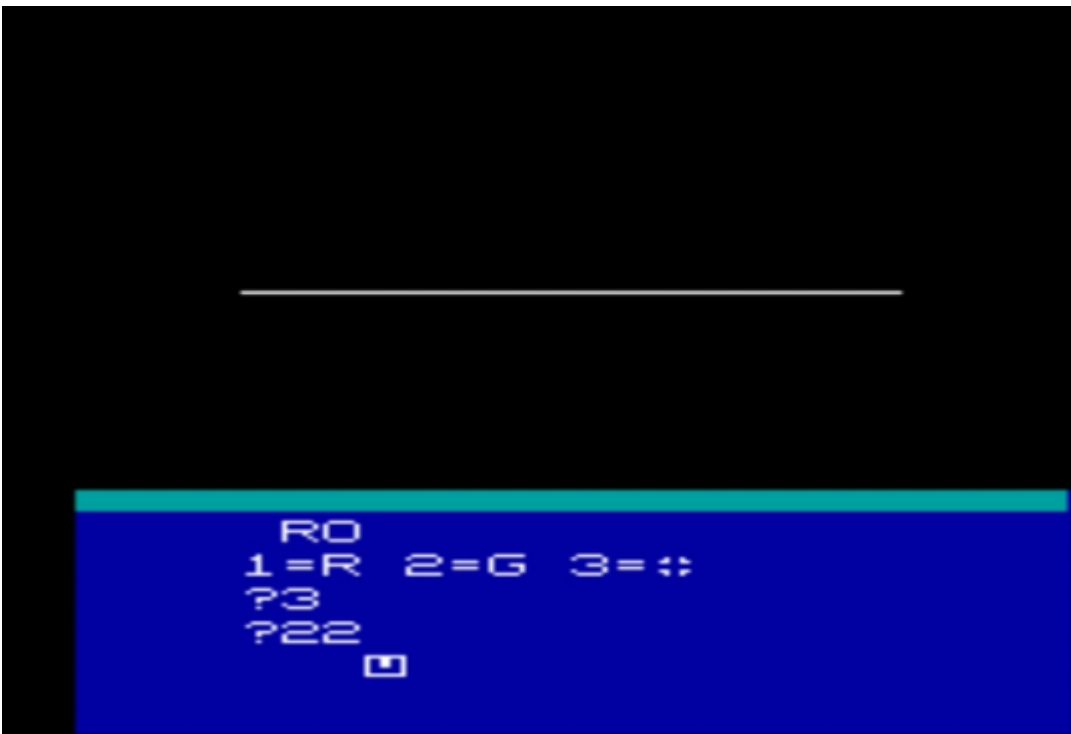
Roulette

First, enter the program. Then, before running, type:

N=0

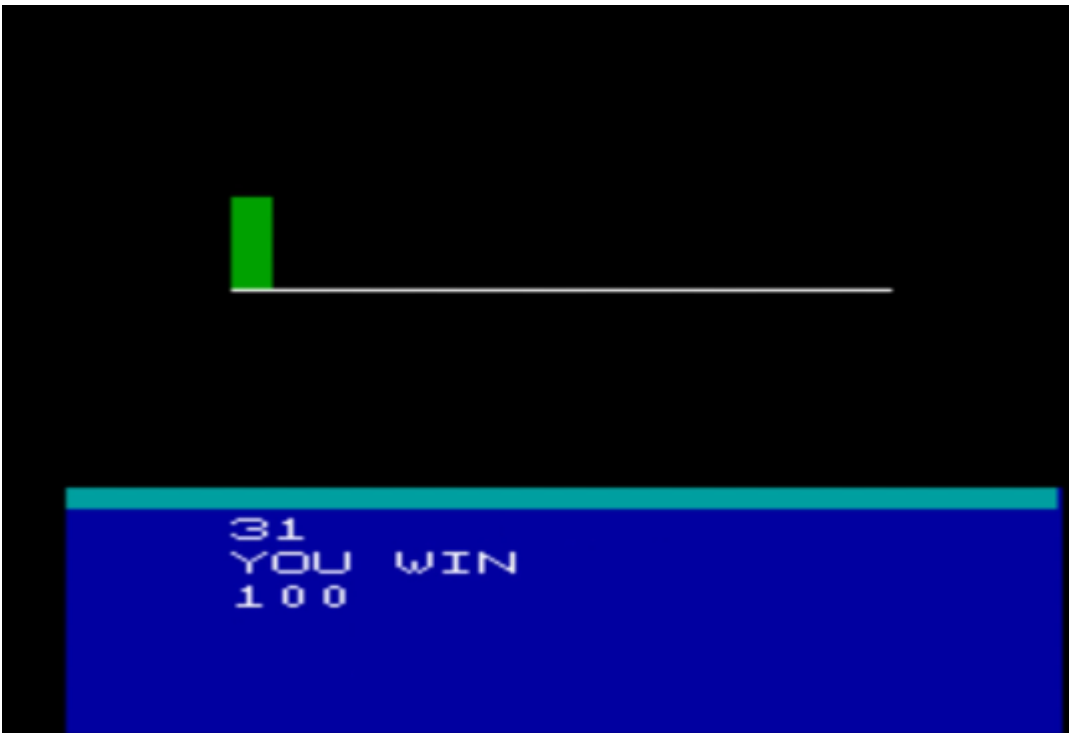
BARH 30

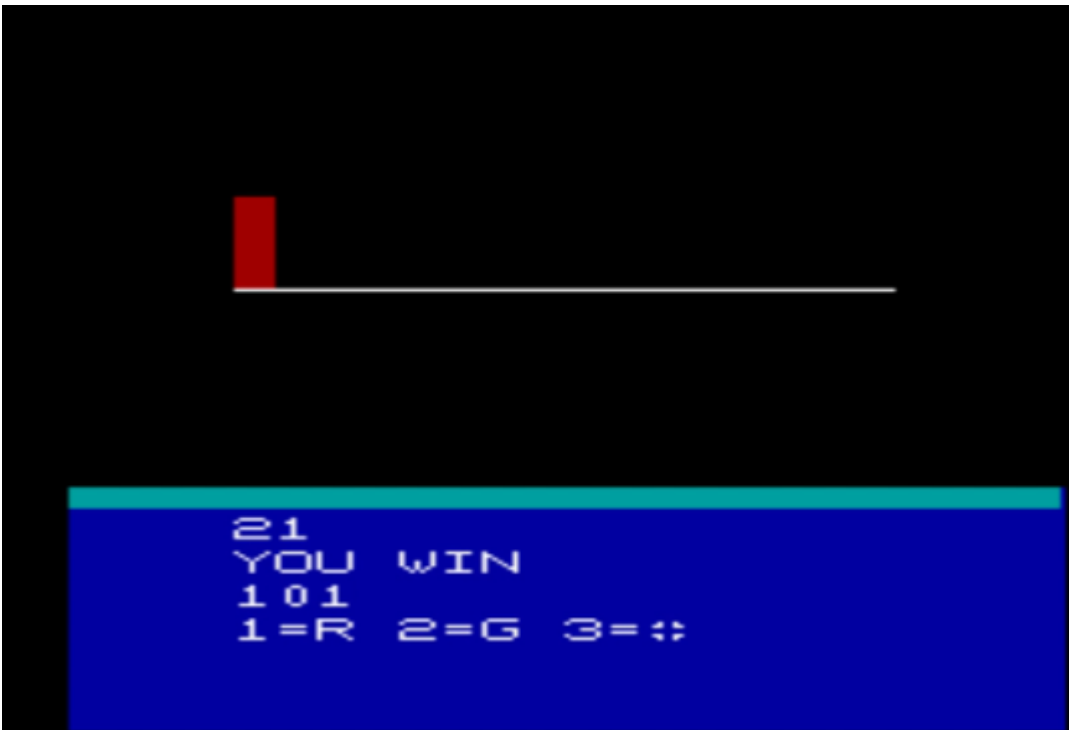
```
PROG R0
M=100
R=1 3 5 7 9 12 14 16 18 19 21 25 27 30 32 34 36
G=2 4 6 8 10 11 13 15 17 20 22 24 26 28 29 31 33 35
WHILE 1
DO
  M=M-1
  "1-R 2-G 3-#"
  P=KEYB
  IF P EQ 3
  THEN N=KEYB
    IF N EQ 99
    THEN EXIT
  ENDI
  S=RANDOM 37
  S=S-1
  S
  IF S EQ 0
  THEN BARC 8
  ELSE
    IF (+/(S EQ R))
    THEN BARC 4
    ELSE BARC 2
  ENDI
  ENDI
  IF (P EQ 3) AND (S EQ N)
  THEN "YOU WIN"
    M=M+35
  ELSE
    IF (P EQ 1) AND (+/(S EQ R))
    THEN "YOU WIN"
      M=M+2
    ELSE
      IF (P EQ 2) AND (+/(S EQ G))
      THEN "YOU WIN"
        M=M+2
      ELSE "YOU LOSE"
    ENDI
  ENDI
  ENDI
M
ENDW
ENDP
```

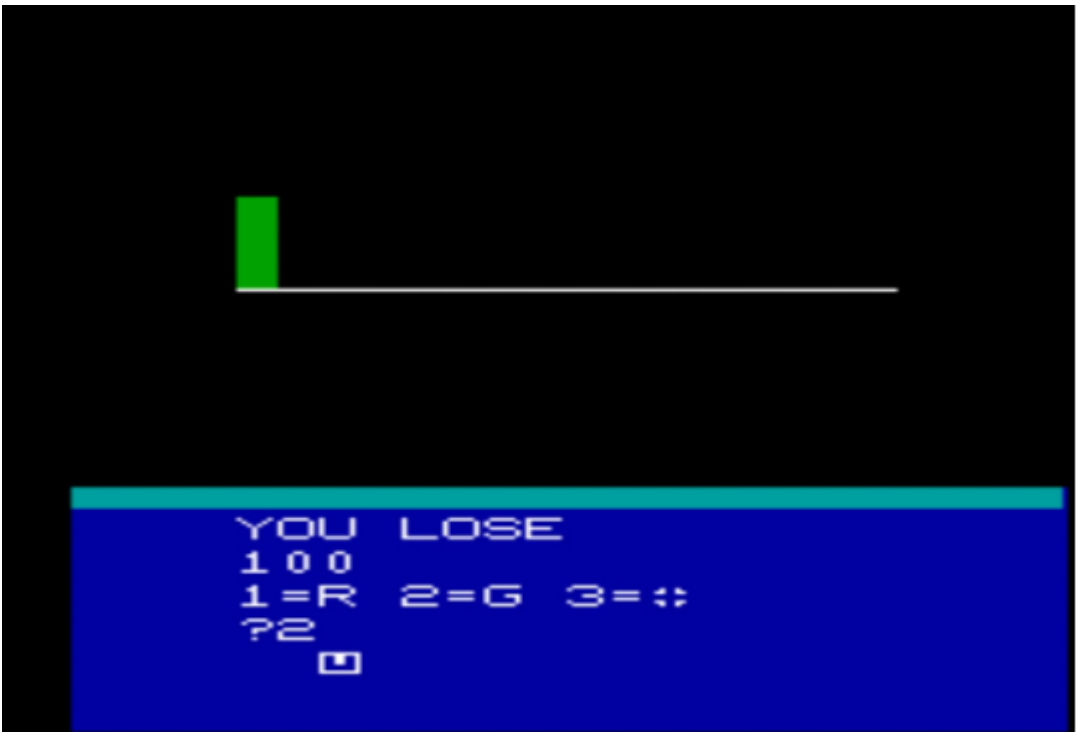
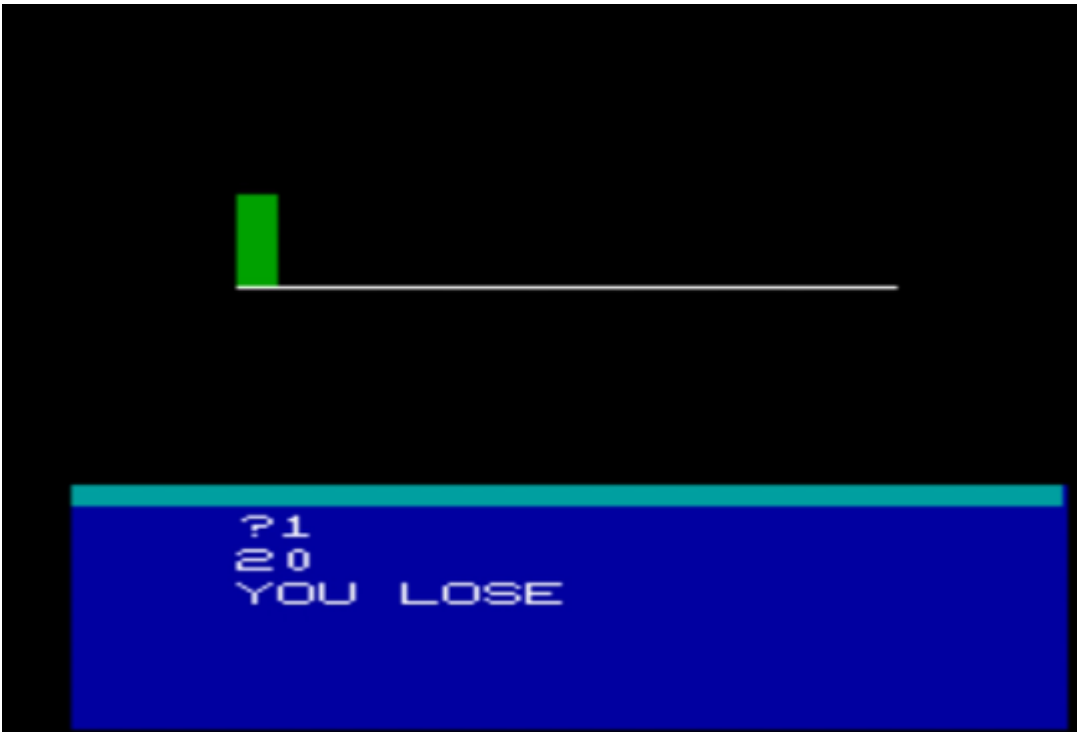


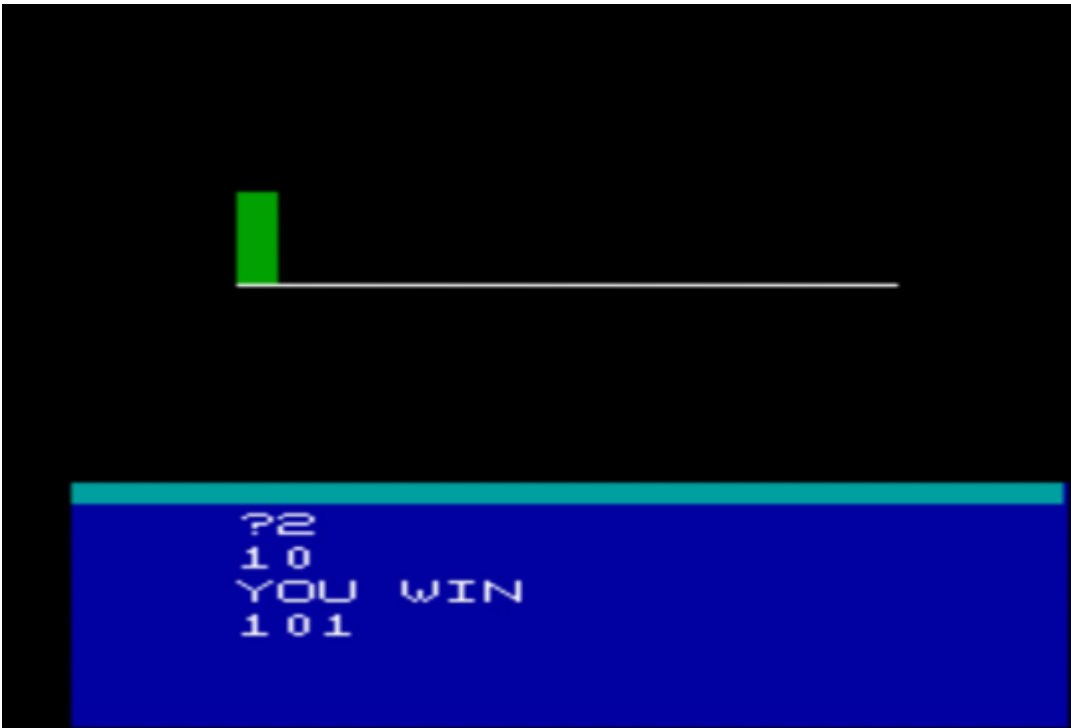
A sample game run.



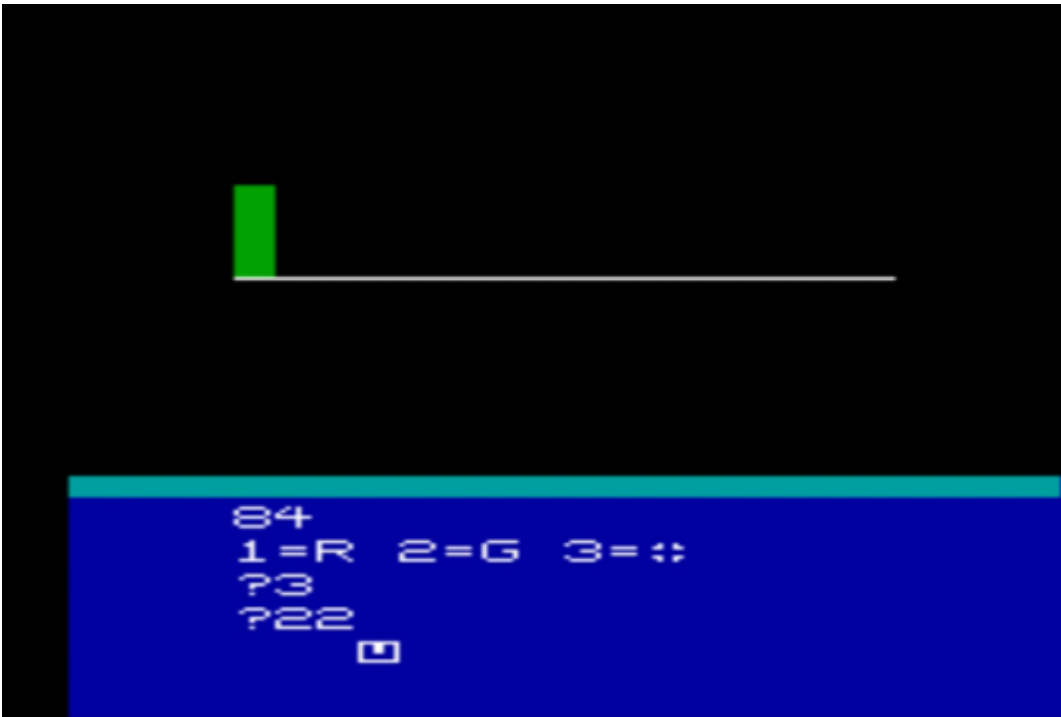




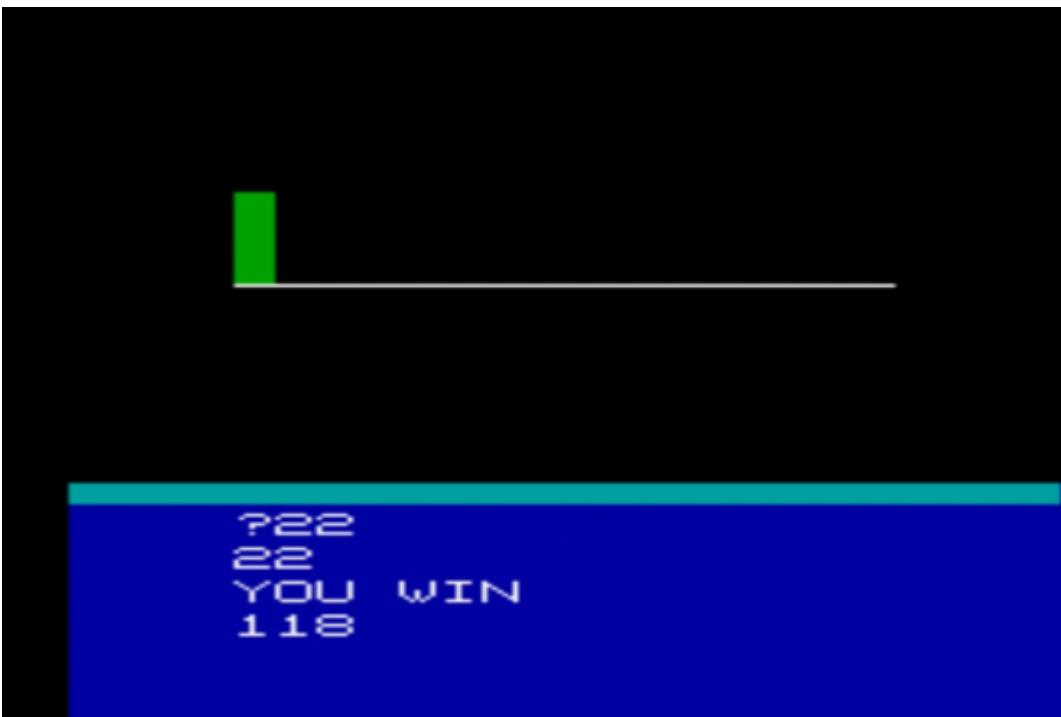


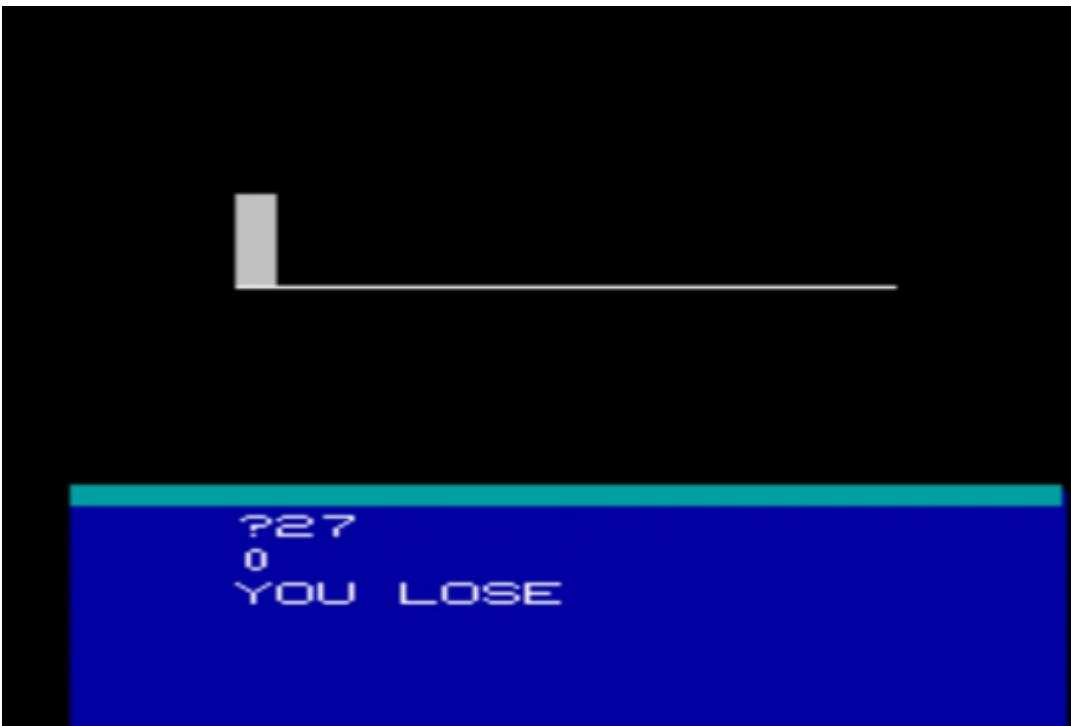
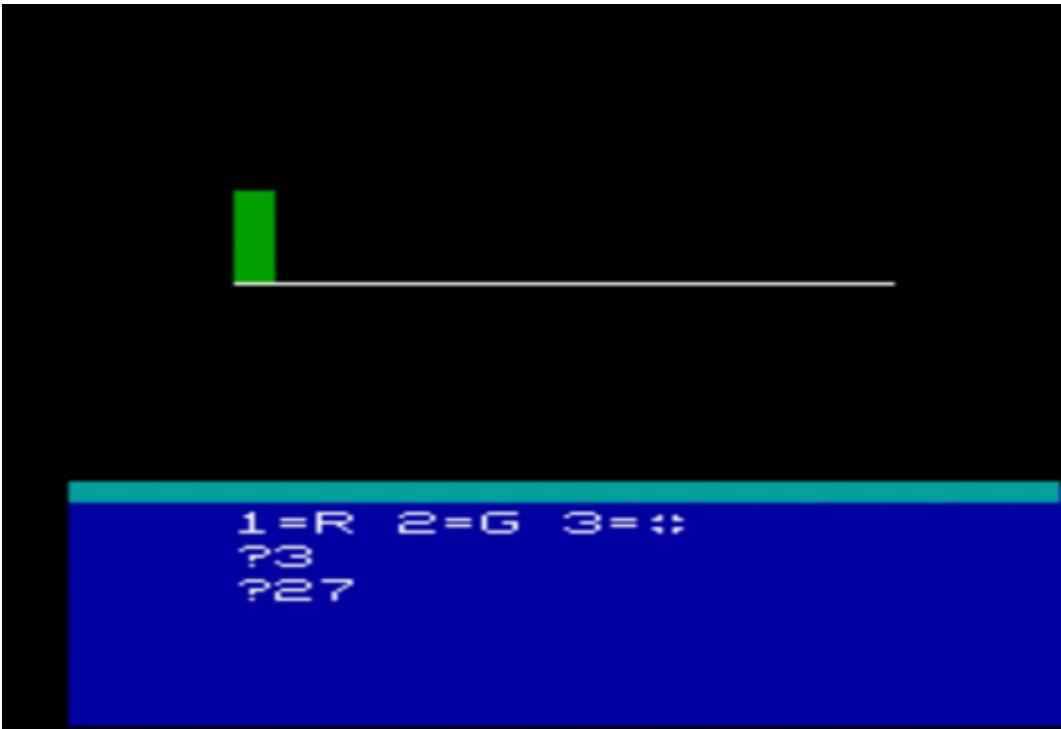


Exit the game by placing a wager on a roll of 99.



A big payout when the ball lands on the number the player wagered on.





The wheel rolls a 0. The house has the edge.

Gunner

A port of David Ahl's 1973 [GUNNER](#) from "101 BASIC Computer Games"

```

PROG GUN
Z=S=S1=0
BARC 10
T=43000-RAND 30000
R=46500
"DISTANCE TO TARGET IS"
T
WHILE 1
DO
    "ELEVATION?"
    B=KEYB
    B2=2×(B÷57.3)
    I=R×(SIN B2)
    X=T-I
    E=FLOOR X
    IF ABS E LT 100
    THEN "YOU WIN"
    EXIT
    ENDI
    IF E GT 100
    THEN "TOO SHORT BY"
        BAR -30
    ELSE
        "TOO FAR BY"
        BAR 30
    ENDI
    ABS E
    S=S+1
    IF S EQ 6
    THEN "YOU LOSE"
    EXIT
    ENDI
ENDW
ENDP

```

GUN
DISTANCE TO TARG
■ET IS

■ET IS
13337
ELEVATION?
?13
□

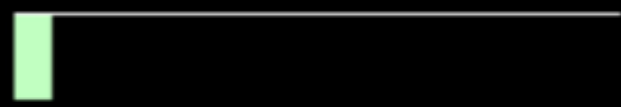


TOO FAR BY
7046
ELEVATION?
?11
□

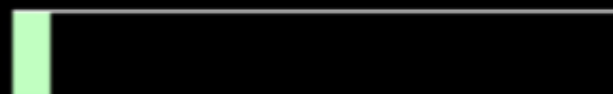


TOO FAR BY
4081
ELEVATION?
?7





TOO SHORT BY
2088
ELEVATION?
?8
□



TOO SHORT BY
520
ELEVATION?
?9
□



TOO FAR BY
1032
ELEVATION?
78.5
□



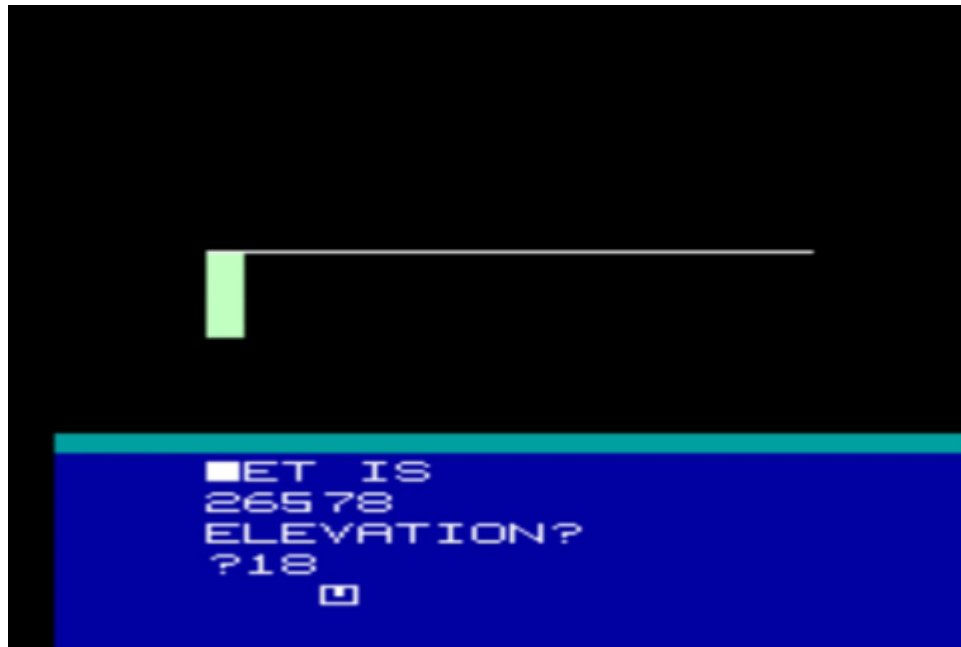
TOO FAR BY
258
YOU LOSE





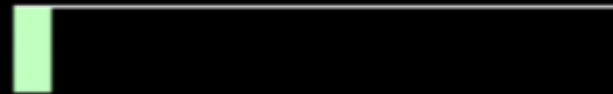
GUN
DISTANCE TO TARG
■ET IS
26578

Another game.

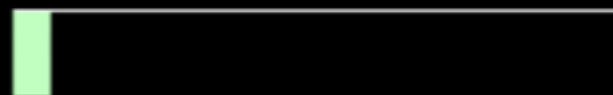




TOO FAR BY
753
ELEVATION?
?17
□



TOO SHORT BY
577
ELEVATION?
?17.5
□



ELEVATION?
?17.5
YOU WIN



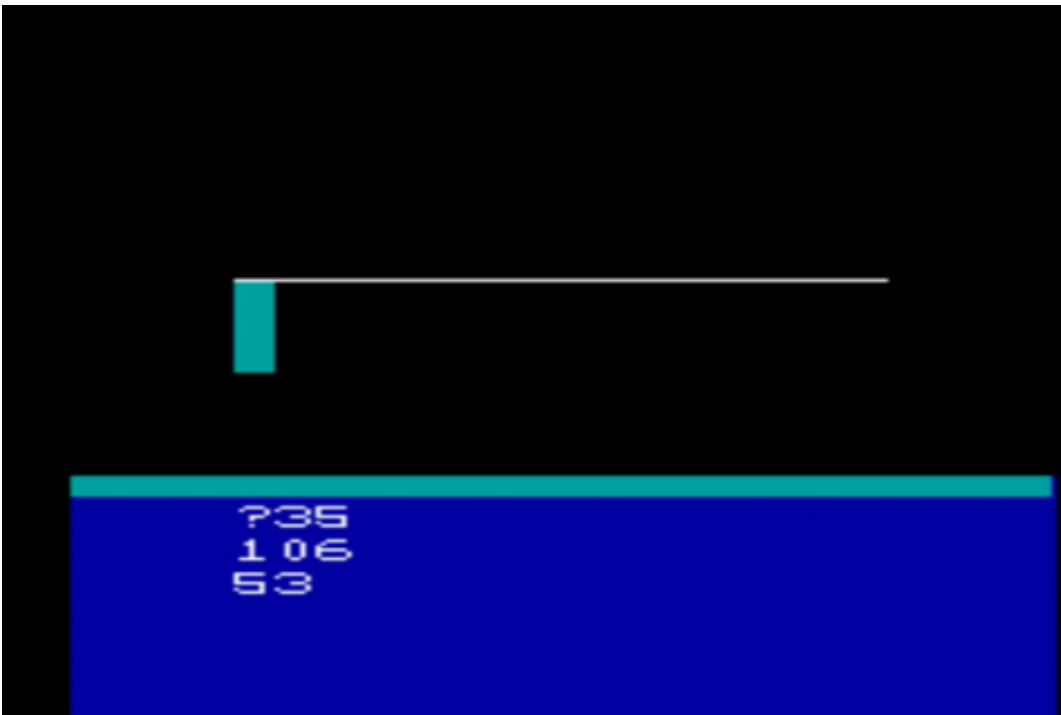
Collatz Bounce

The Collatz conjecture is a long-standing open question in mathematics built around an extremely simple process. Starting from any positive integer, a sequence is generated by repeatedly applying one of two rules: if the current value is even, it is divided by two; if it is odd, it is multiplied by three and increased by one.

Despite the simplicity of these operations, no proof exists that the process always ends the same way. The conjecture asserts that, regardless of the starting number, the sequence will eventually reach the value 1, after which it falls into a repeating cycle. This claim has been verified for vast ranges of integers, but a general proof, or a counterexample, remains unknown.

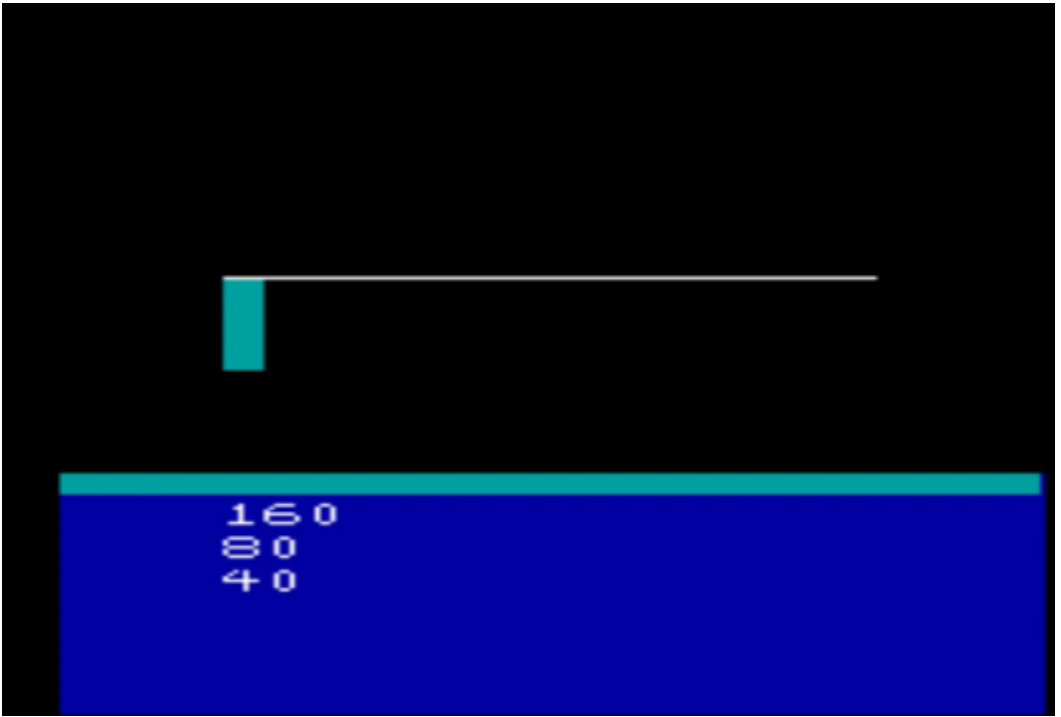
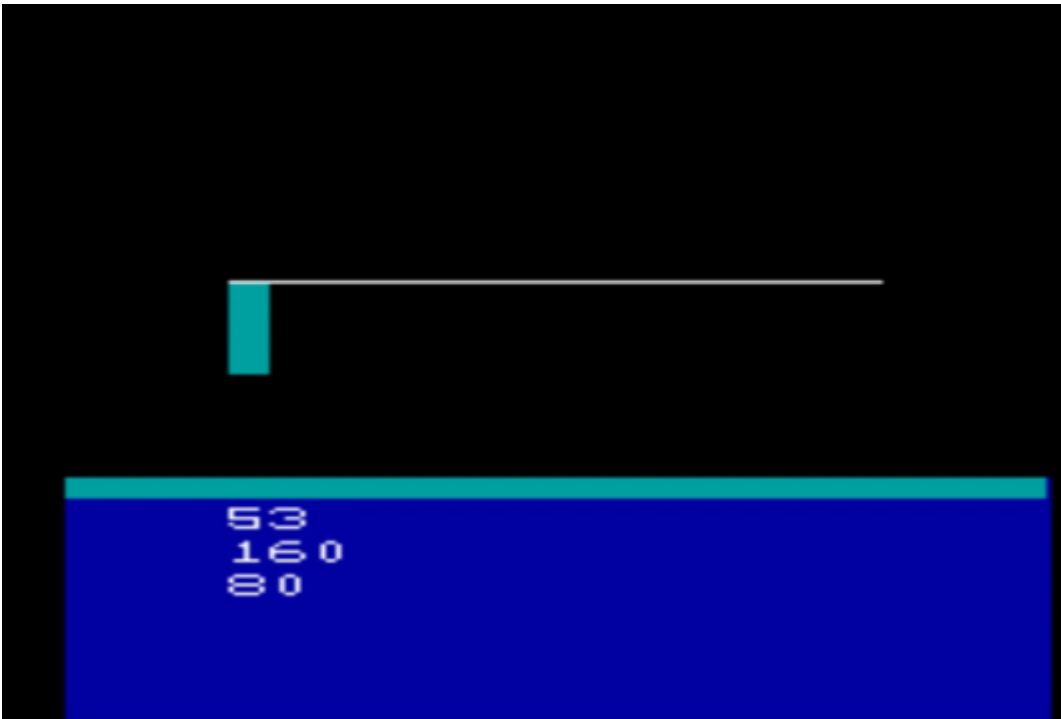
```
PROG COLLATZ
BARC 3
SCORE=0
"ENTER YOUR NUMBER TO COLLATZ BOUNCE"
NUM=I=KEYB
TOP=I
WHILE I NE 1
DO
    IF I MOD 2 EQ 0
    THEN I=I÷2
        BARH -30
    ELSE I=1+3×I
        BARH 30
    ENDI
    SCORE=SCORE+1
    I
    IF I GT TOP
    THEN TOP=I
    ENDI
ENDW
BARH 0
"NUM TOP SCORE"
NUM,TOP,SCORE
ENDP
```

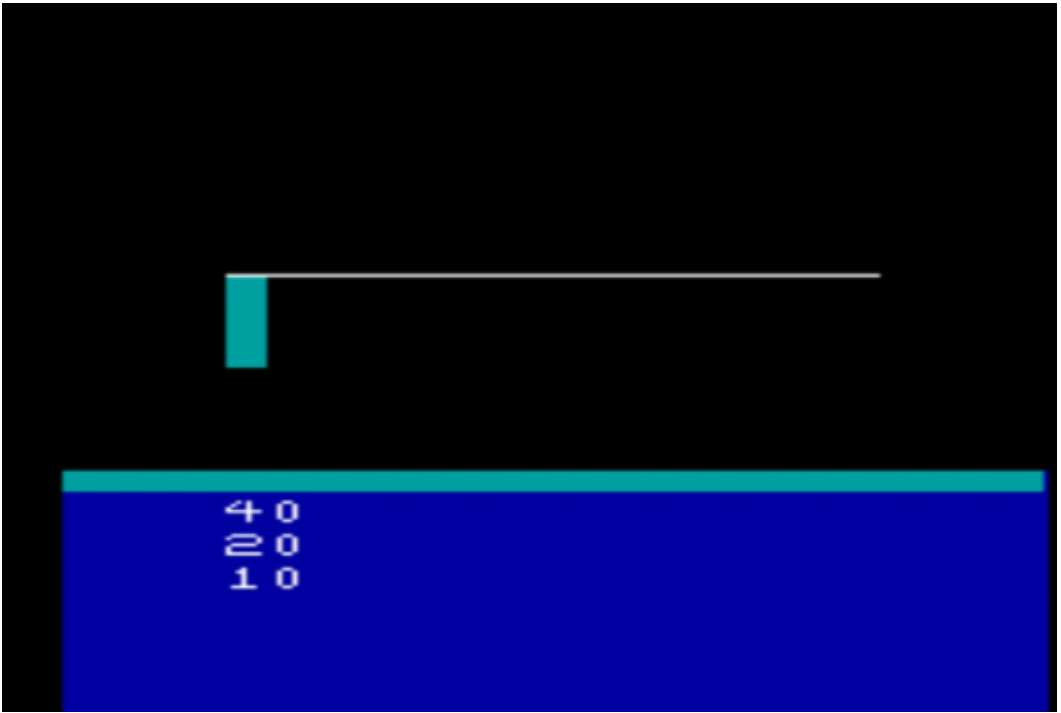
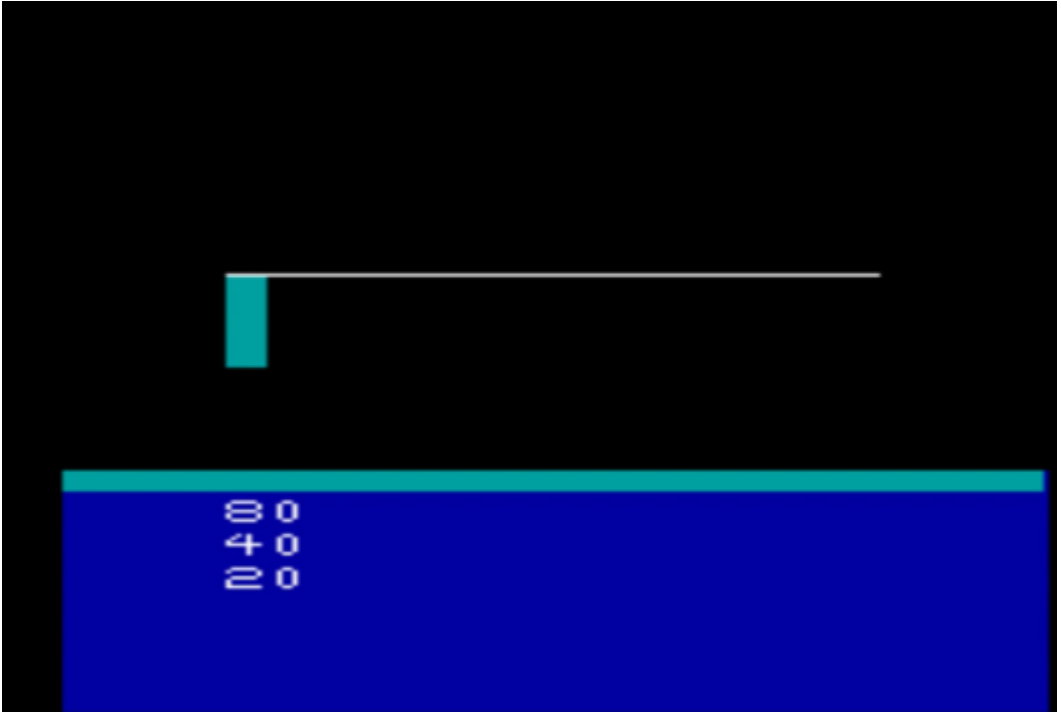
C
ENTER YOUR NUMBE
R TO COLLATZ BO
UNCE

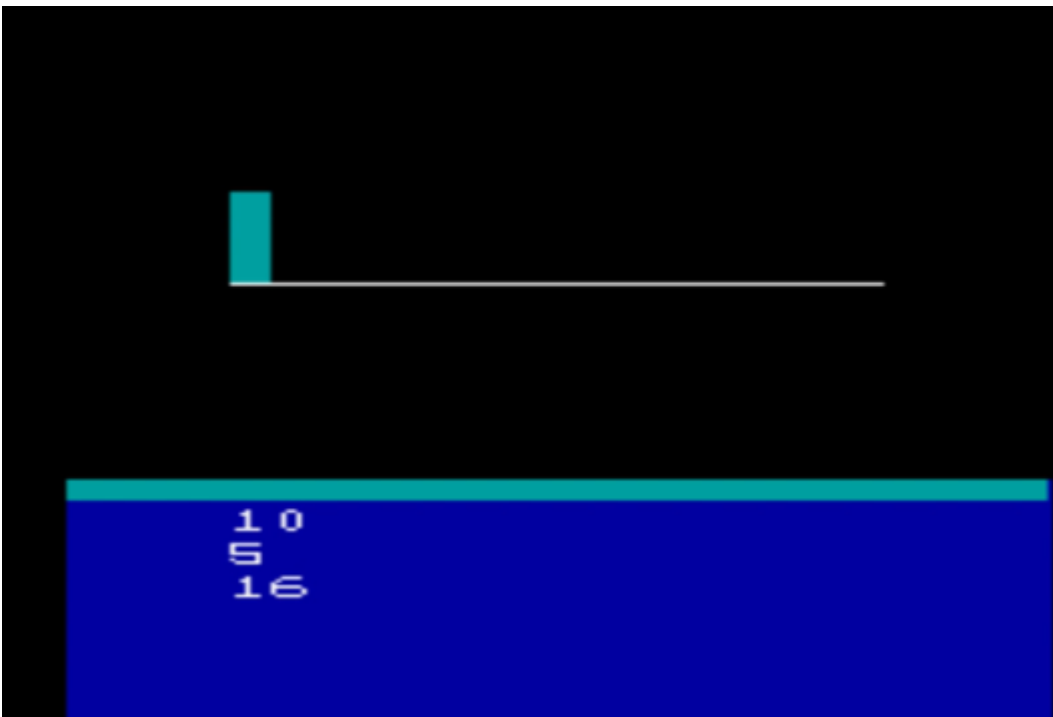
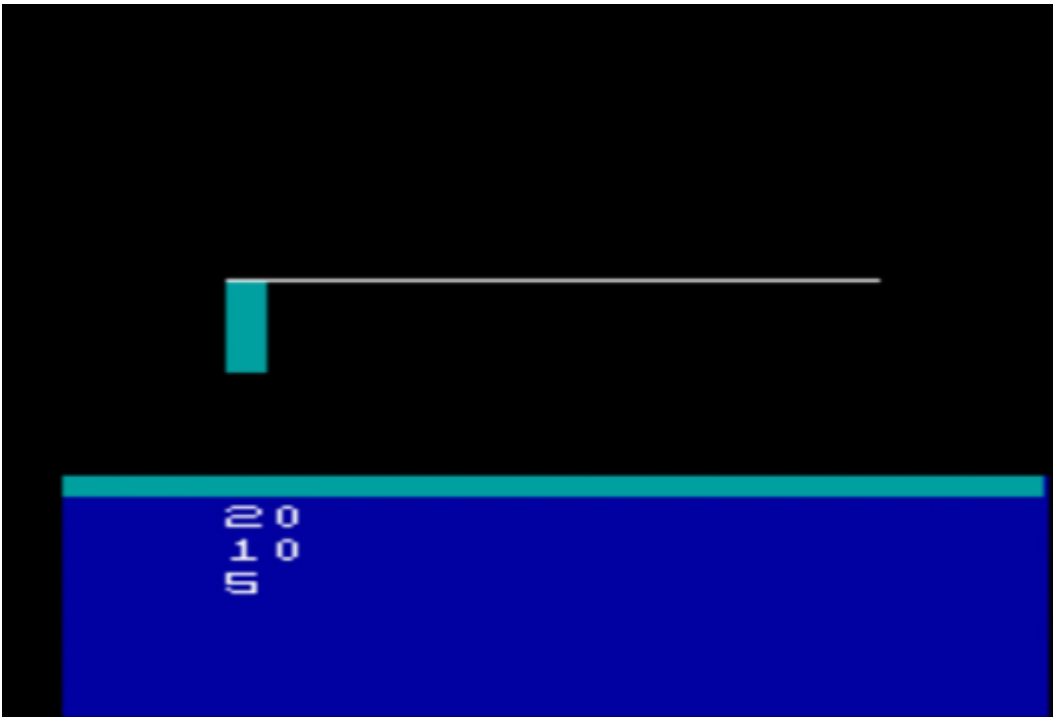


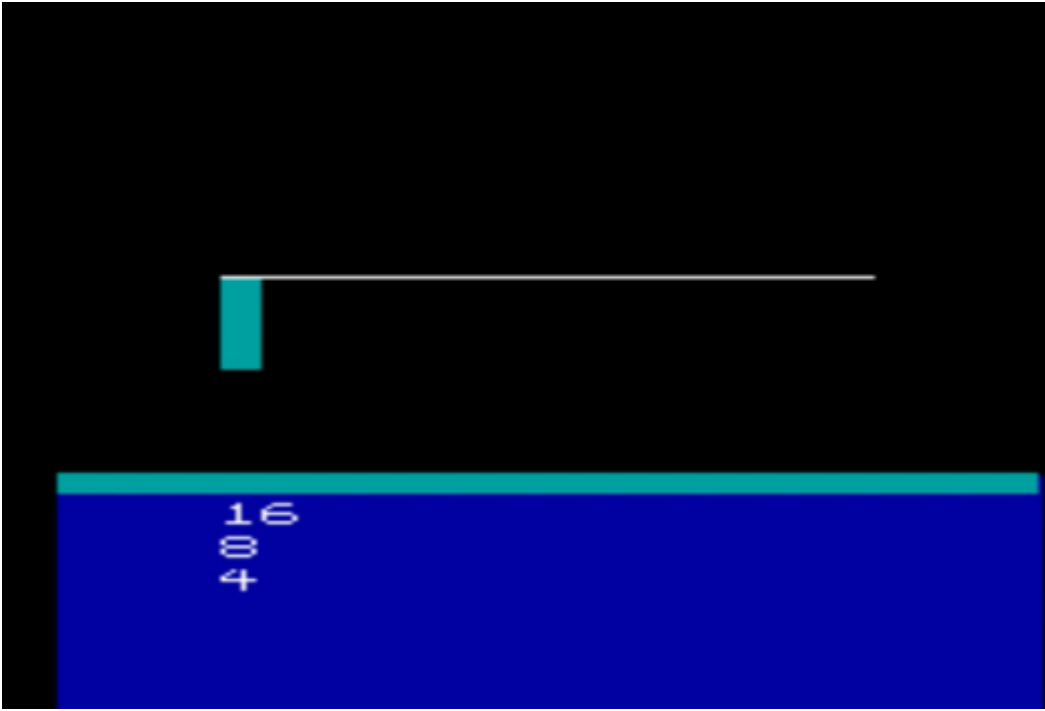
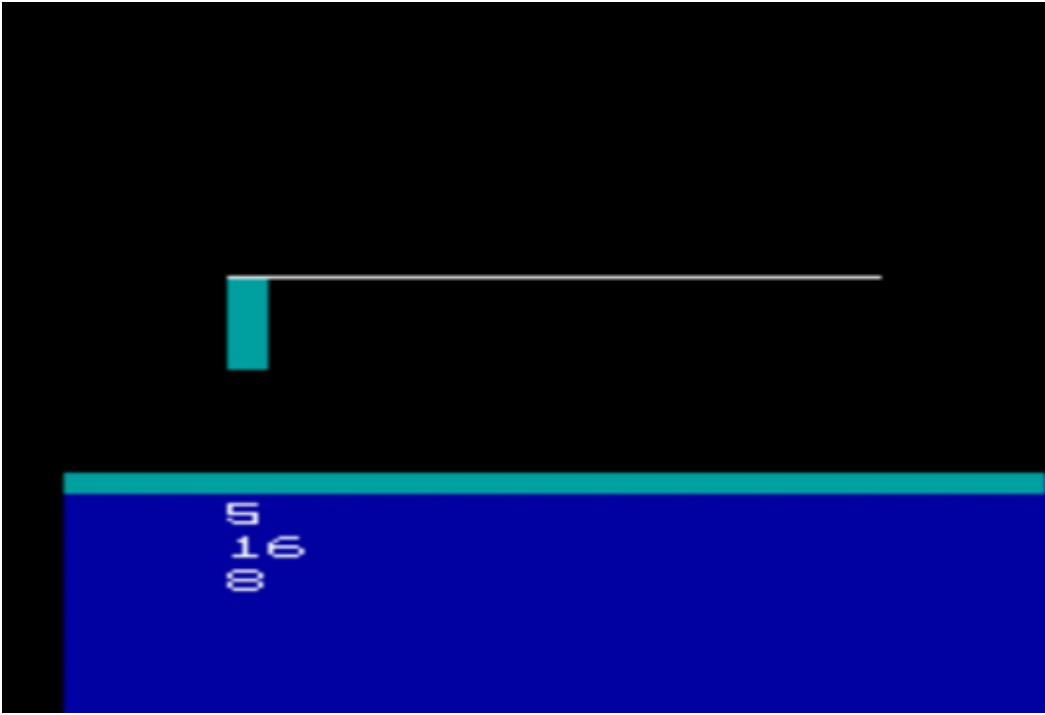


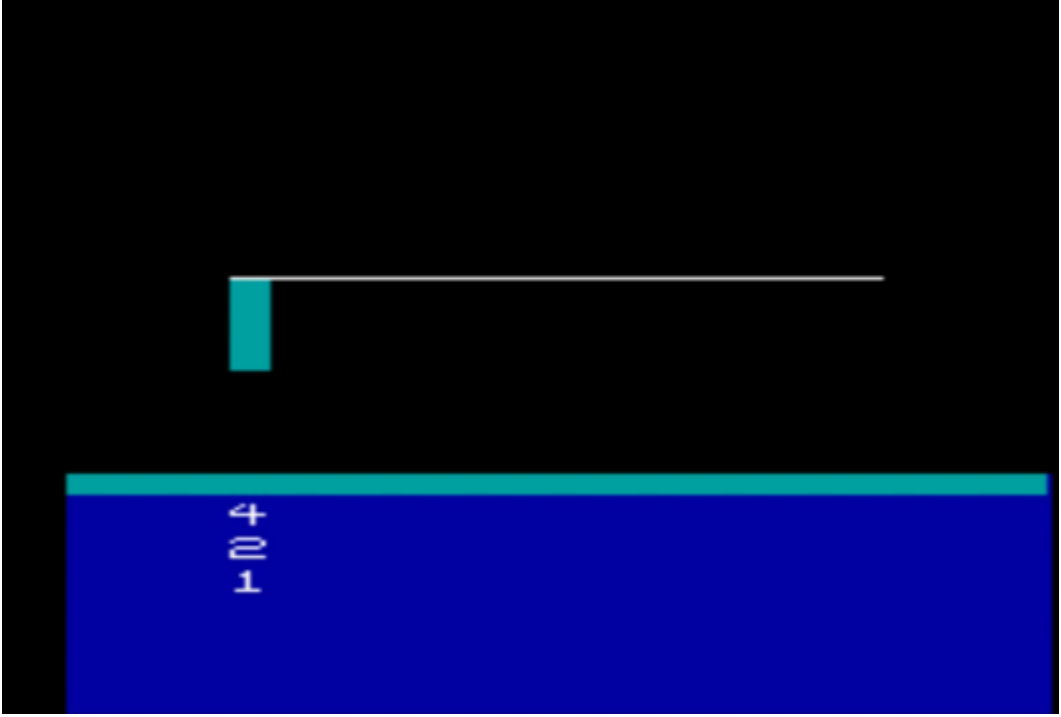
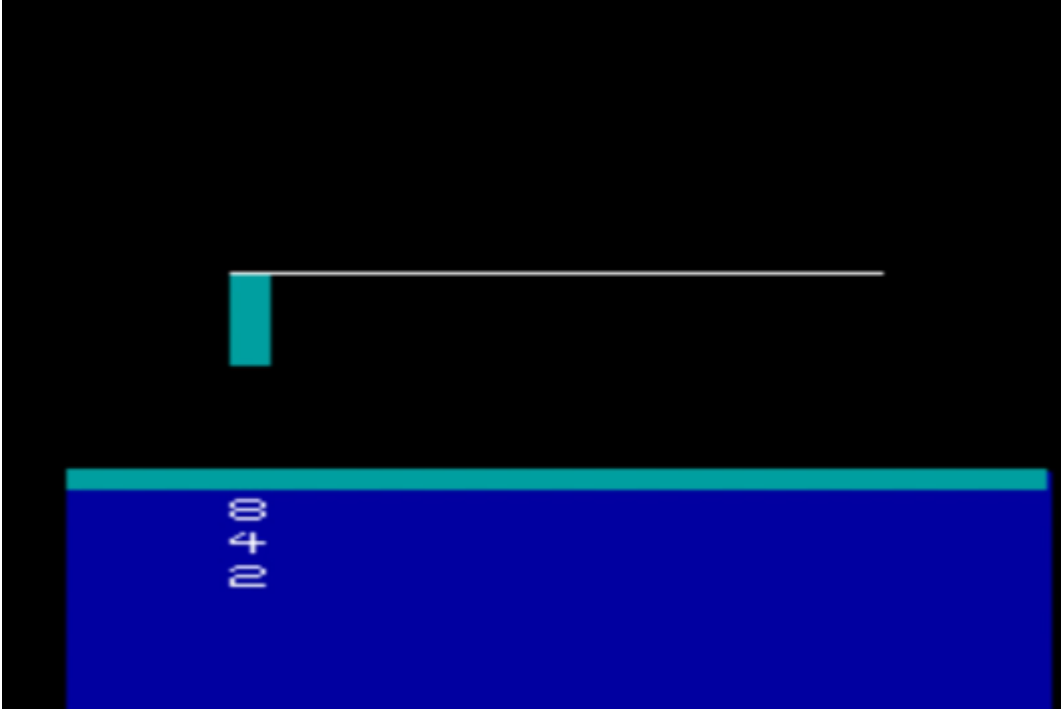
106
53
160

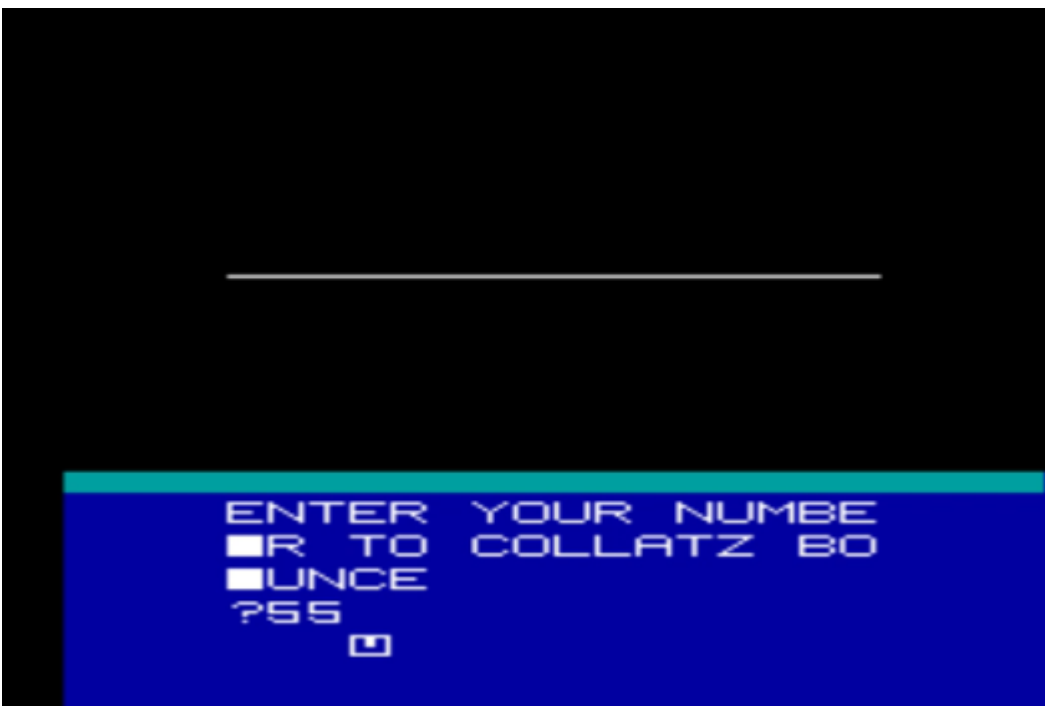
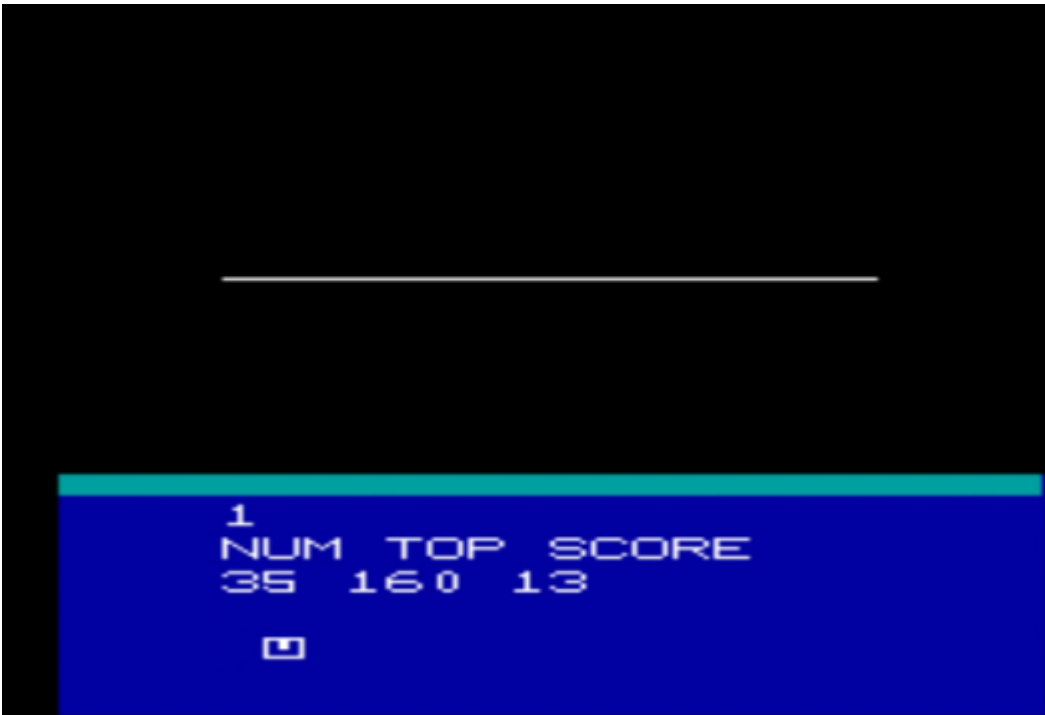




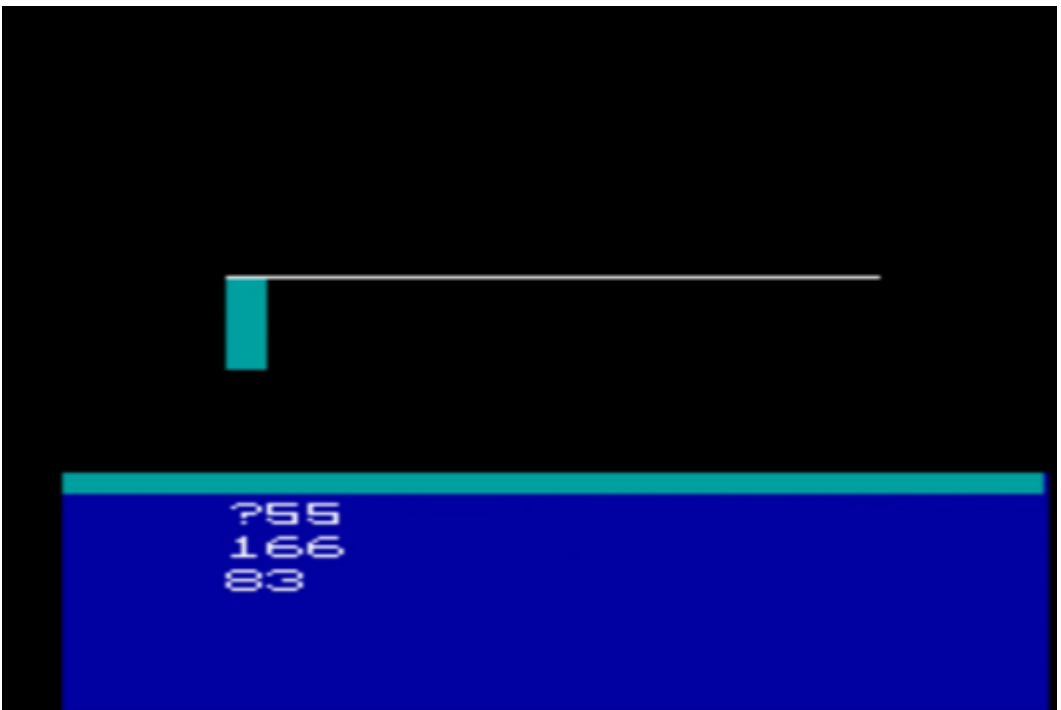
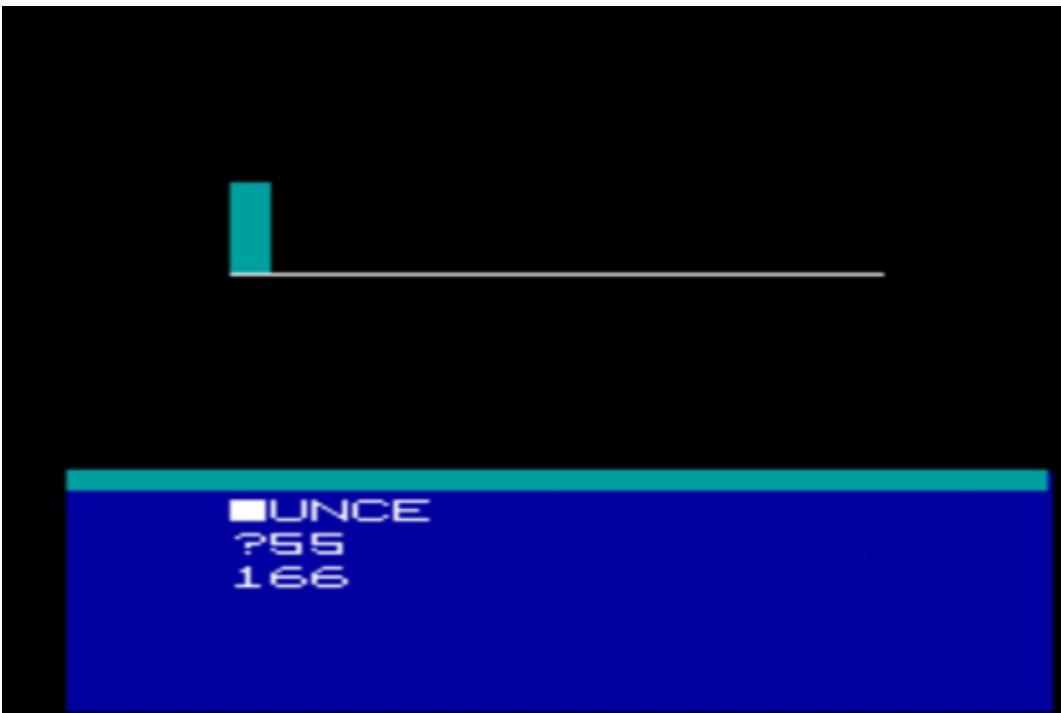






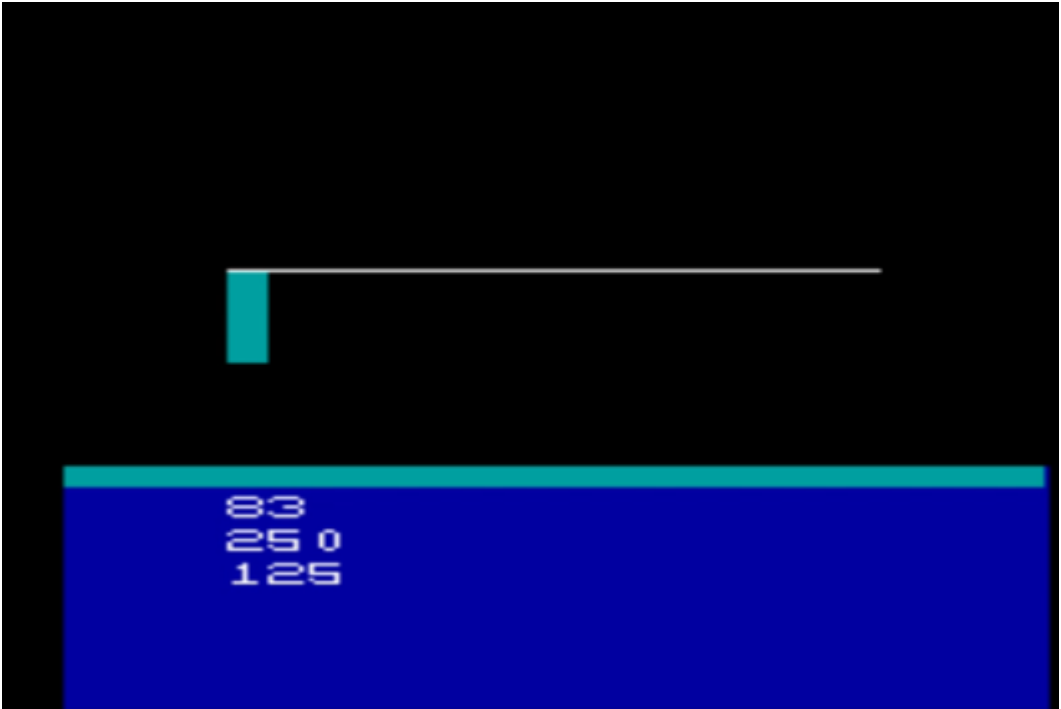


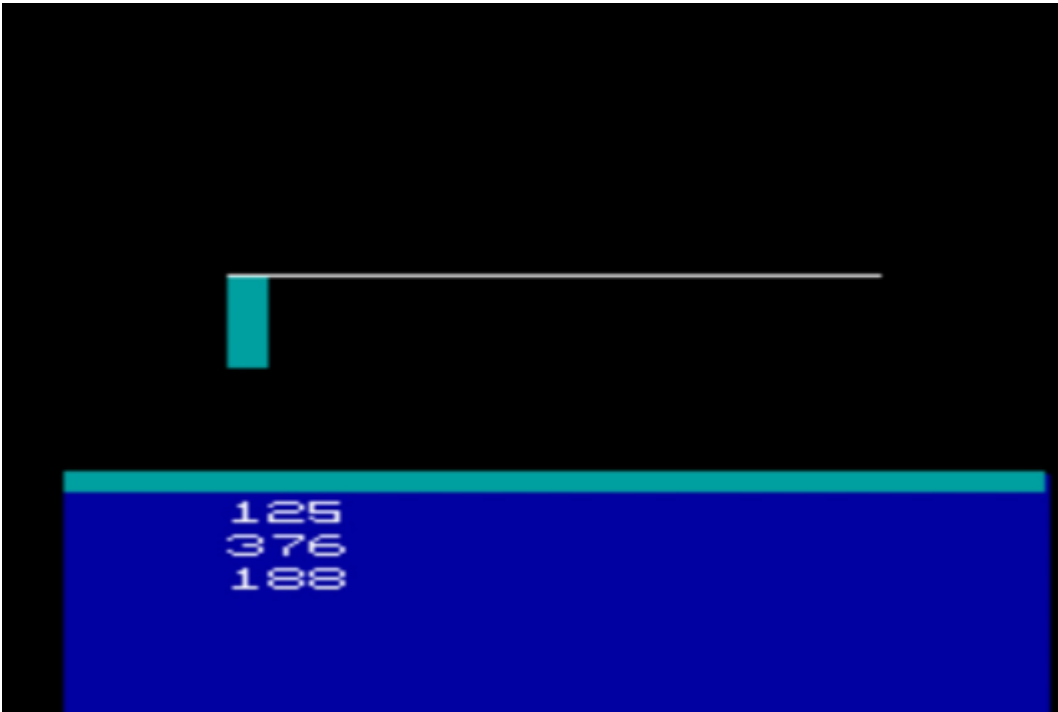
Another run, starting at 55.

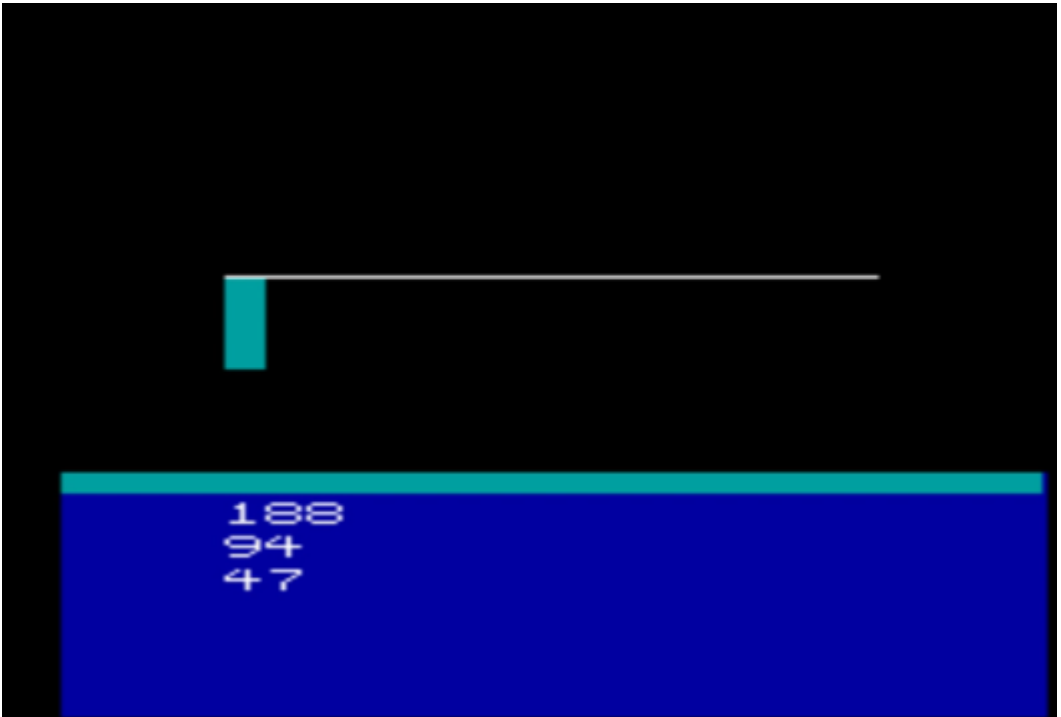
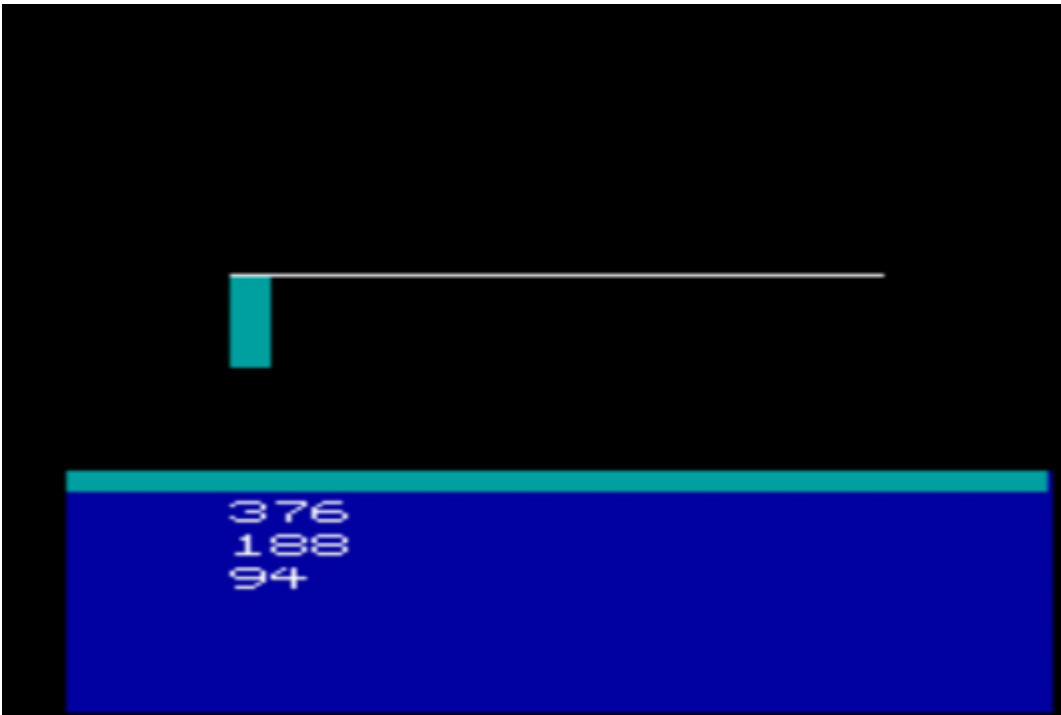


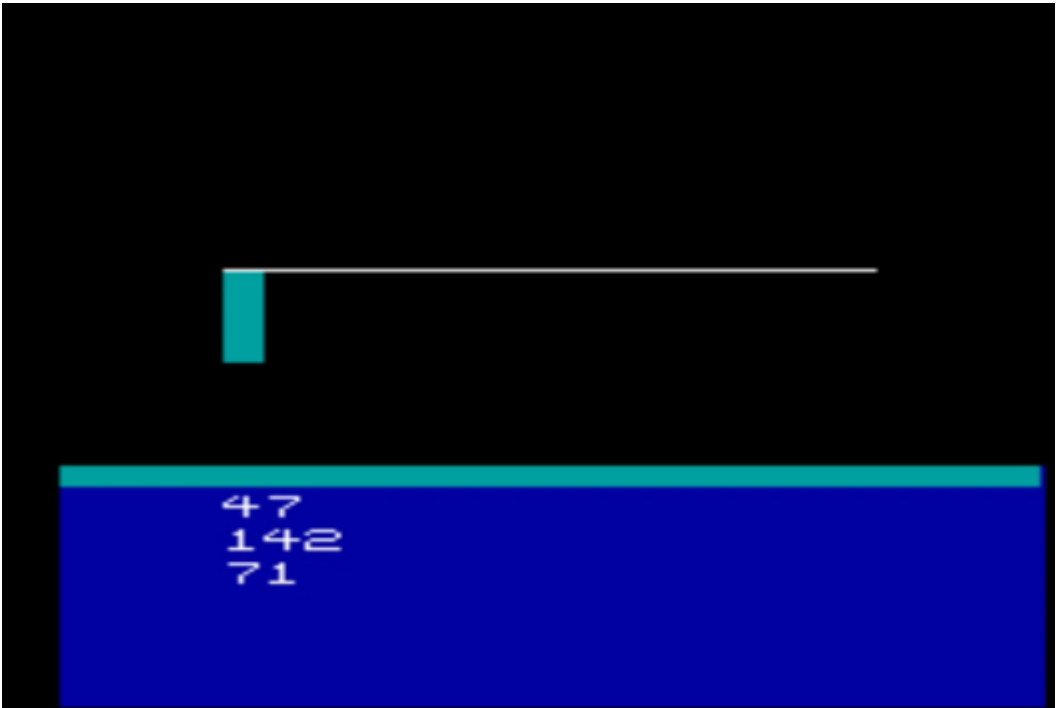
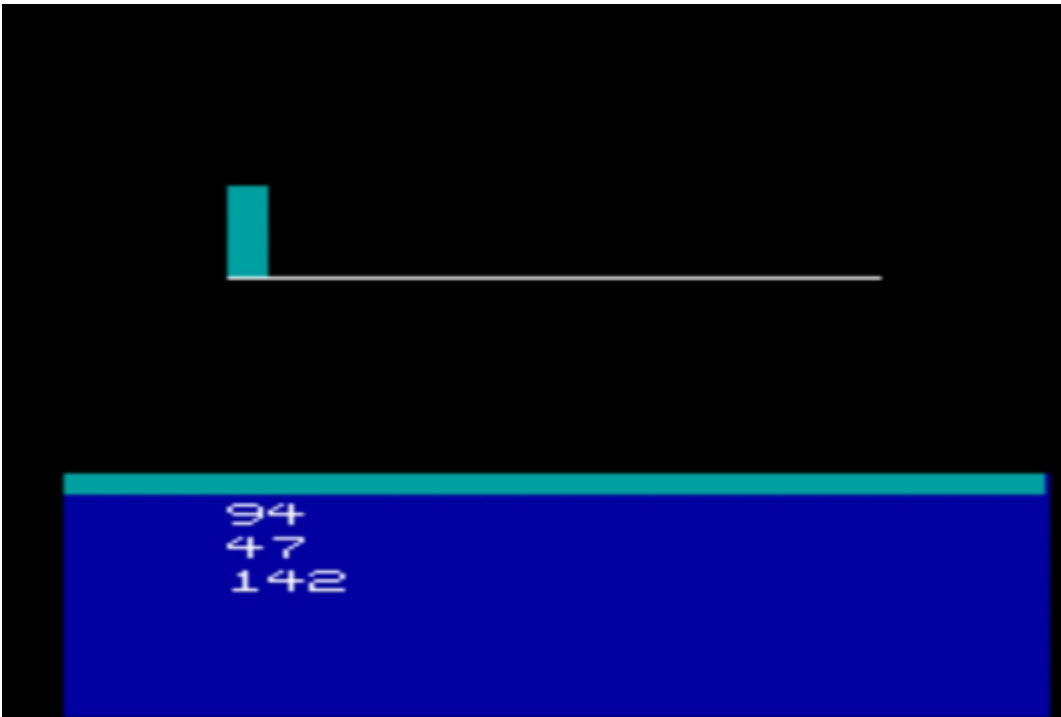


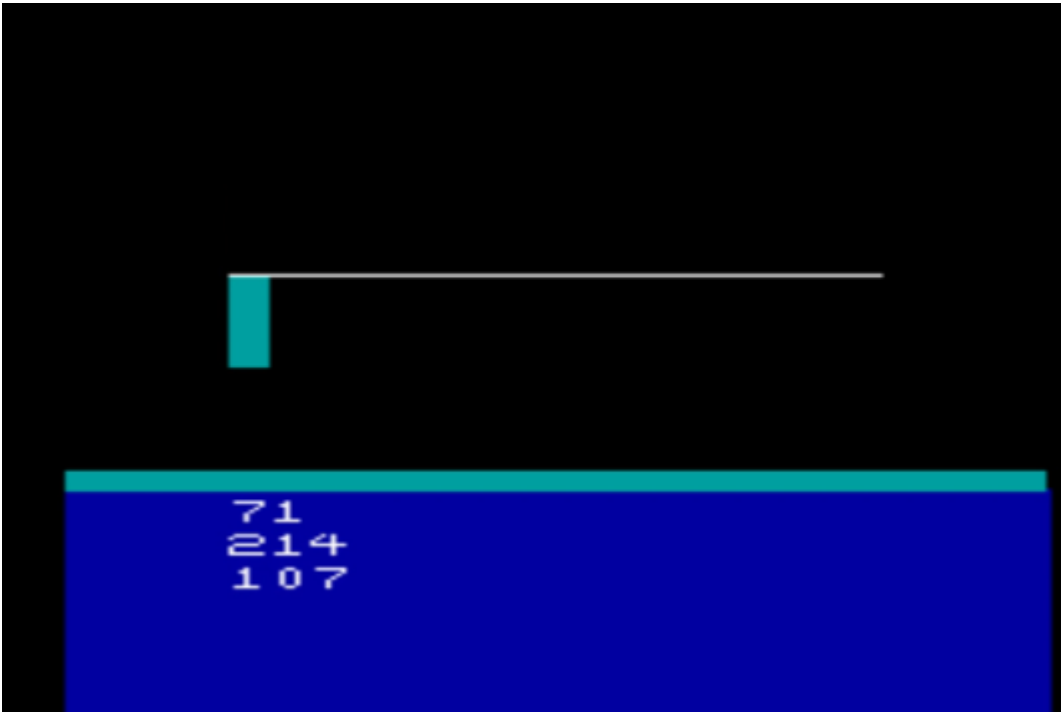
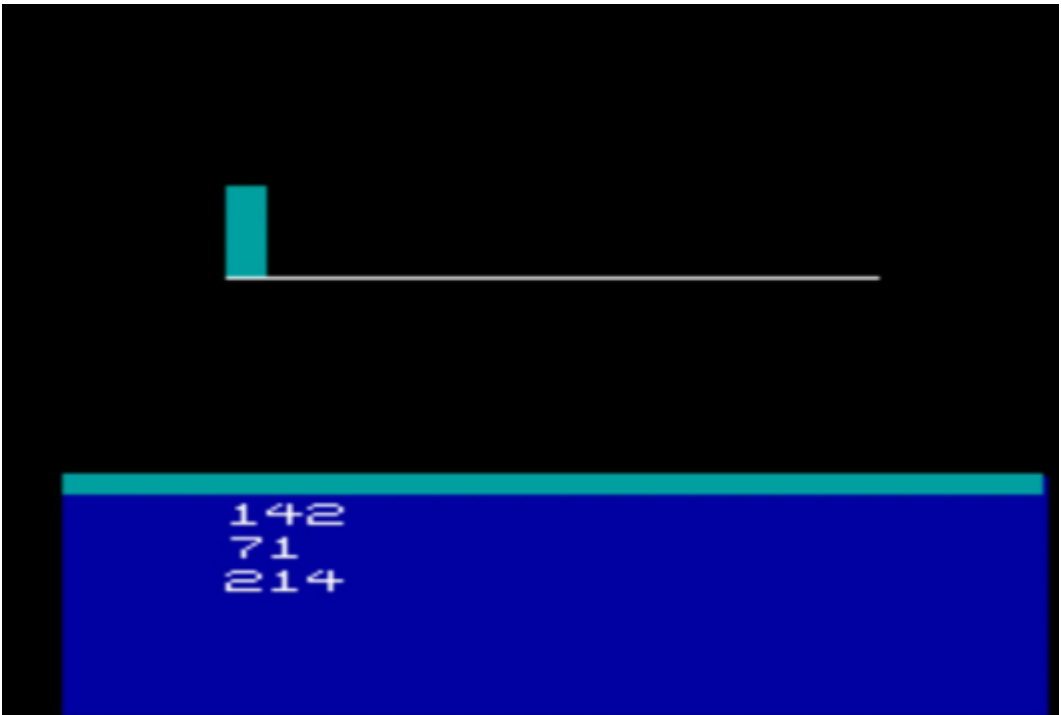
166
83
25 0

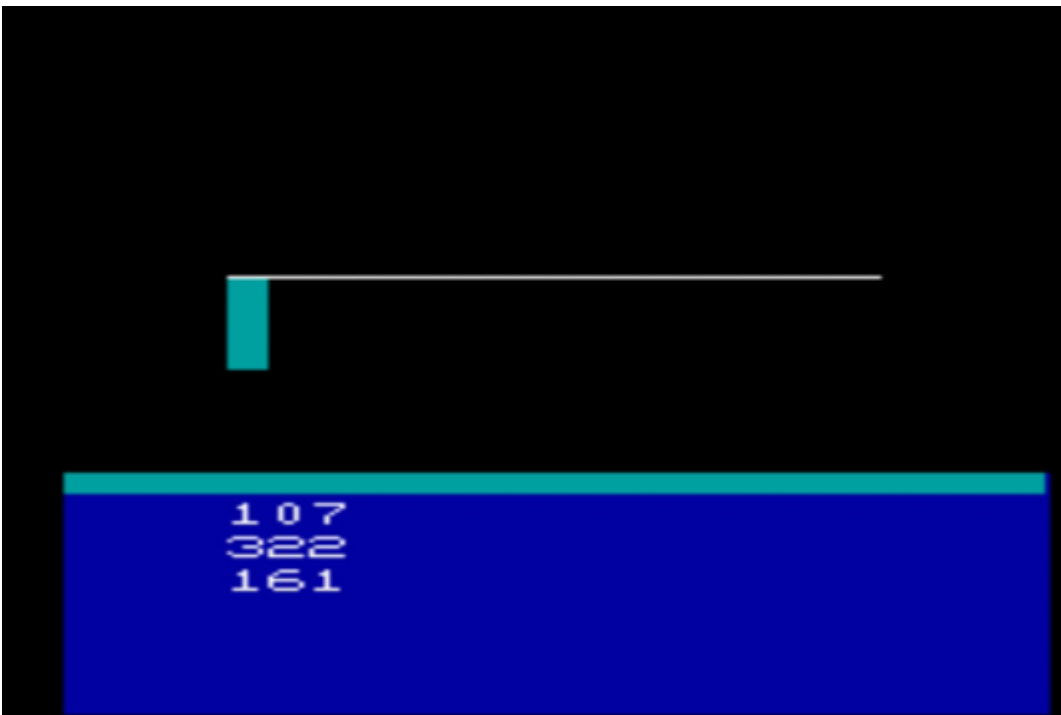
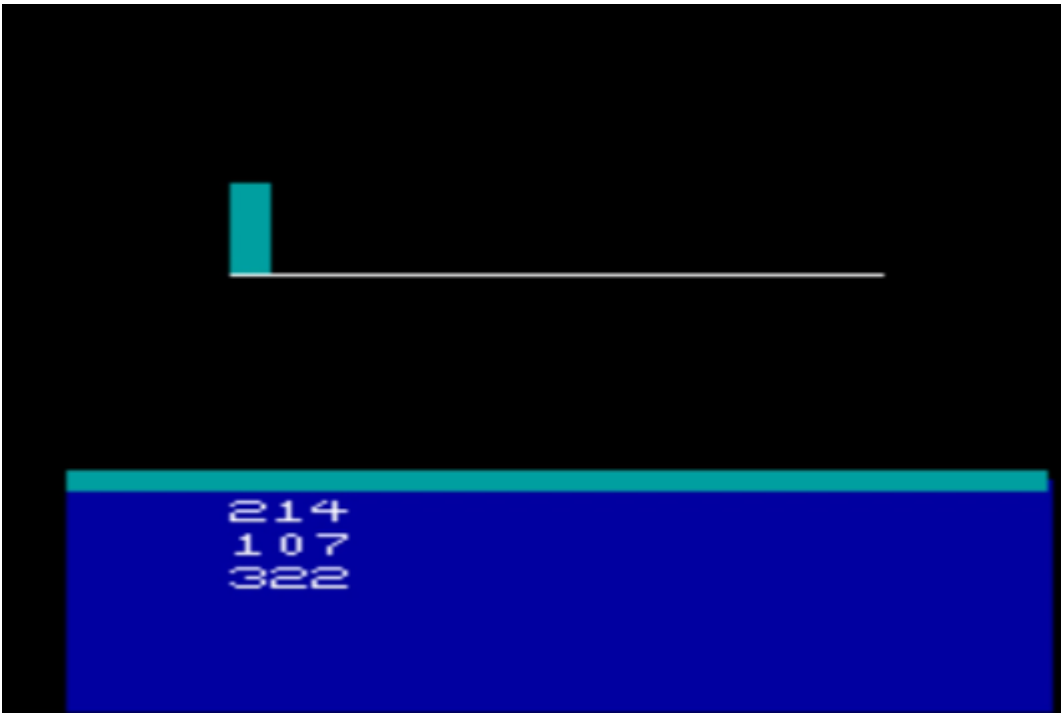


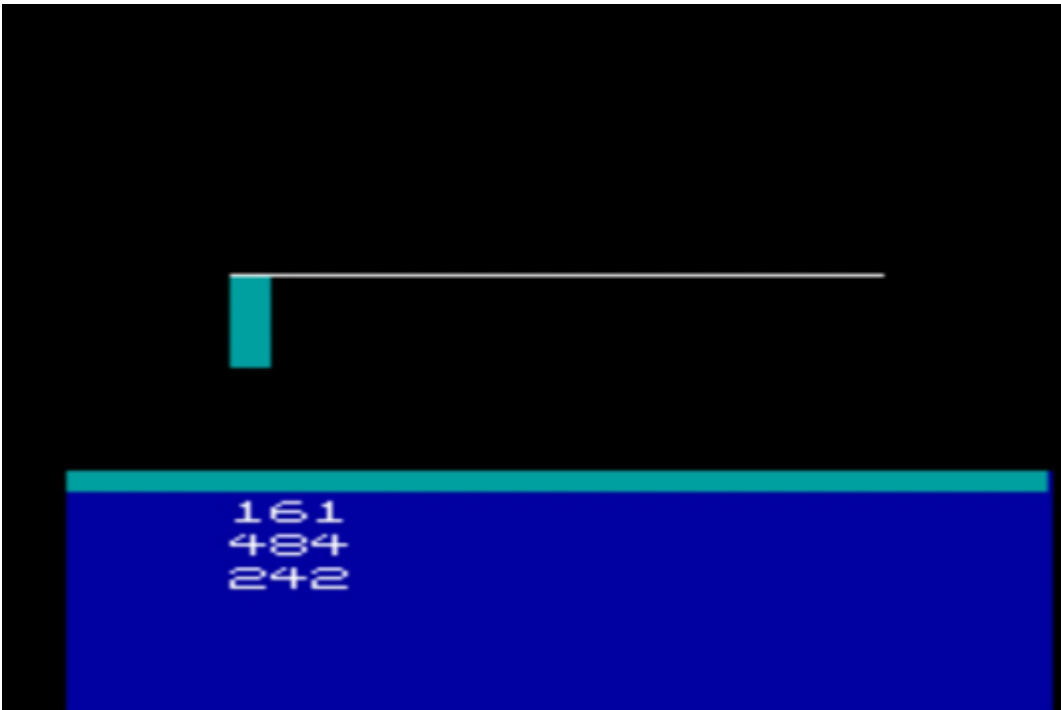
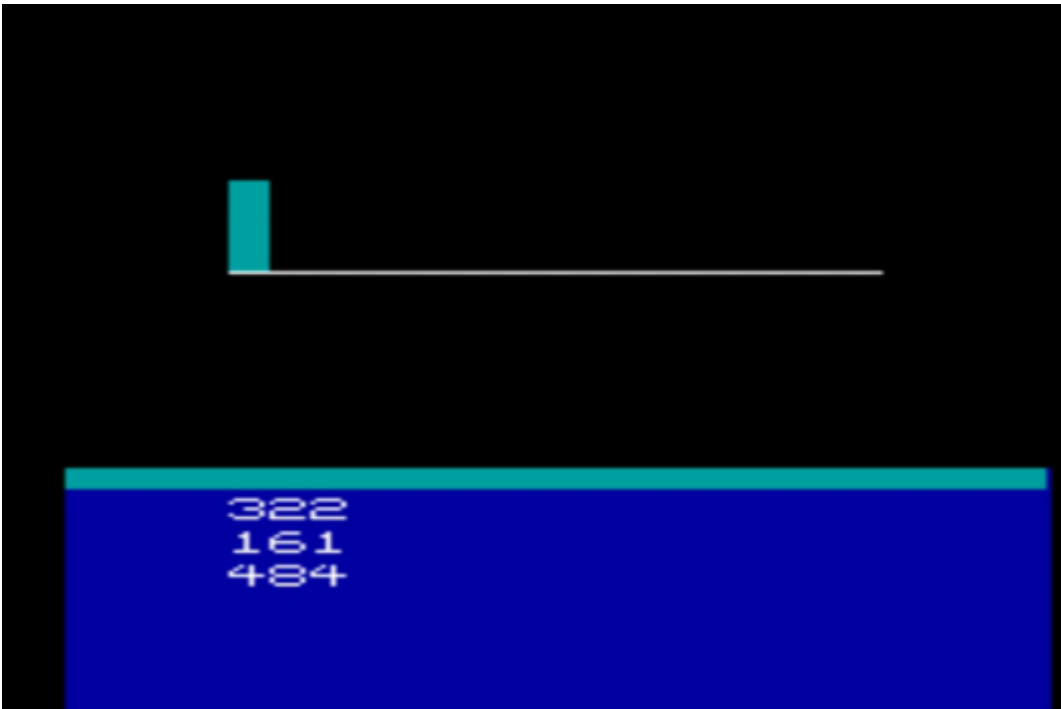










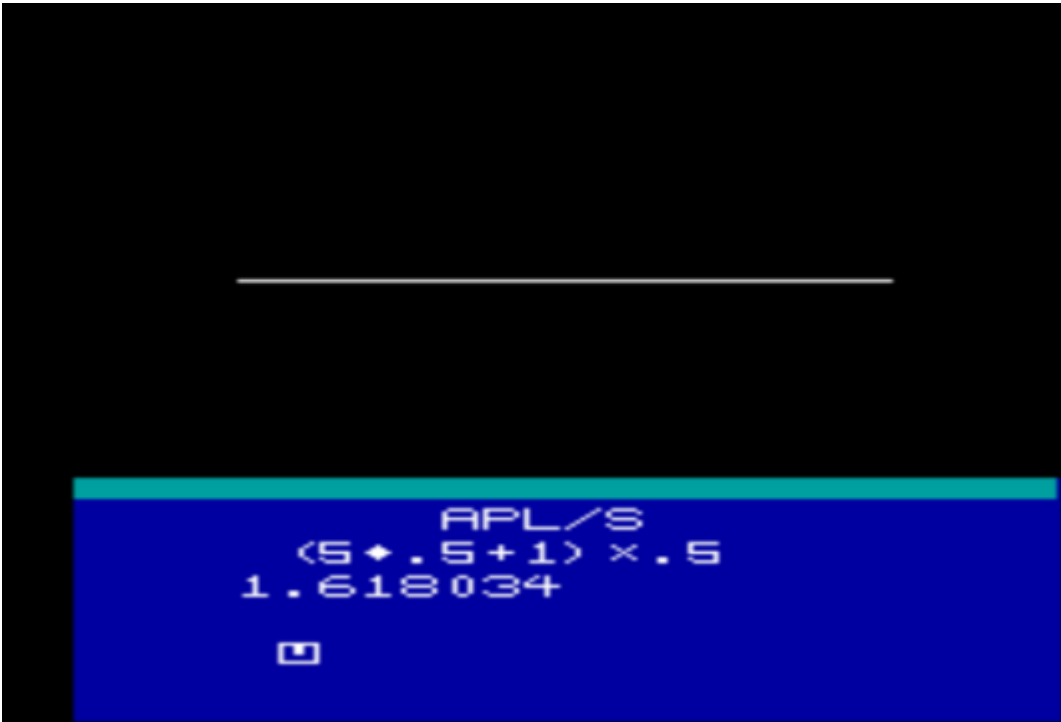


1	NUM	TOP	SCORE
55	9232	112	



A maximum value of 9232 is reached, before settling back to 1 after over a hundred bounces.

Phi and Fibonacci One-Liners



The value of phi, 1.618034, is easy to calculate. It can then be used to generate the Fibonacci sequence.

```
APL/S
FLOOR (.5 + ((1.61
■8 INDEX 5) ÷ (5 ■.
■5) > )
□
```

```
■8♦INDEX 5) ÷ (5♦.  
■5) > >  
1 1 2 3 5  
□
```

```
FLOO  (0.5+( (PHI=
■ ( (Z=5♦0.5) +1) ×0
■ .5) ♦INDE 16) ÷Z)
■
```

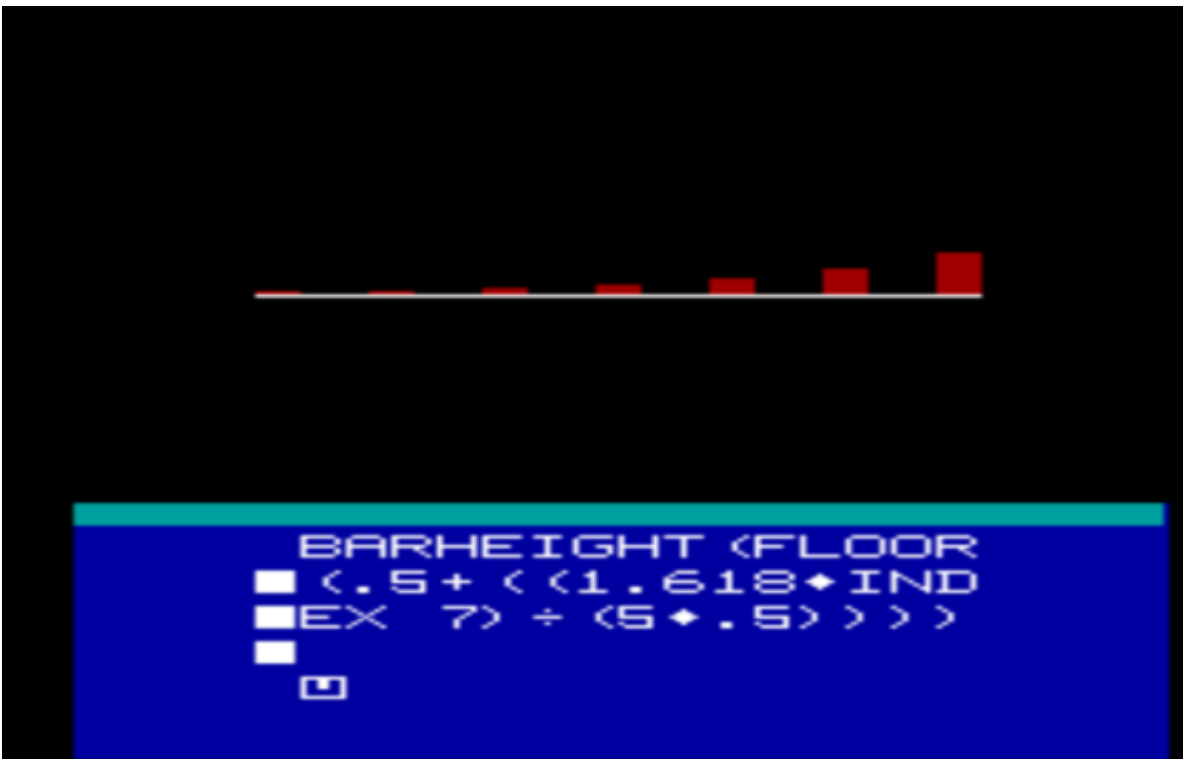


```
1 1 2 3 5 8 13 2
■1 34 55 89 144
■233 377 610 987
```



FLOOR (.5+((PHI=((Z=5*.5)+1) × .5) * INDEX 5) ÷ Z).

The formula for the Fibonacci sequence, using Phi, as a one-liner.



Using the barchart to graph the sequence.



Add a little color. See the [APL MOOC](#) to see this done in a modern APL.

```
X=Y=1
```

```
PROG FIB
```

```
Z=X+Y
```

```
Z
```

```
X=Y
```

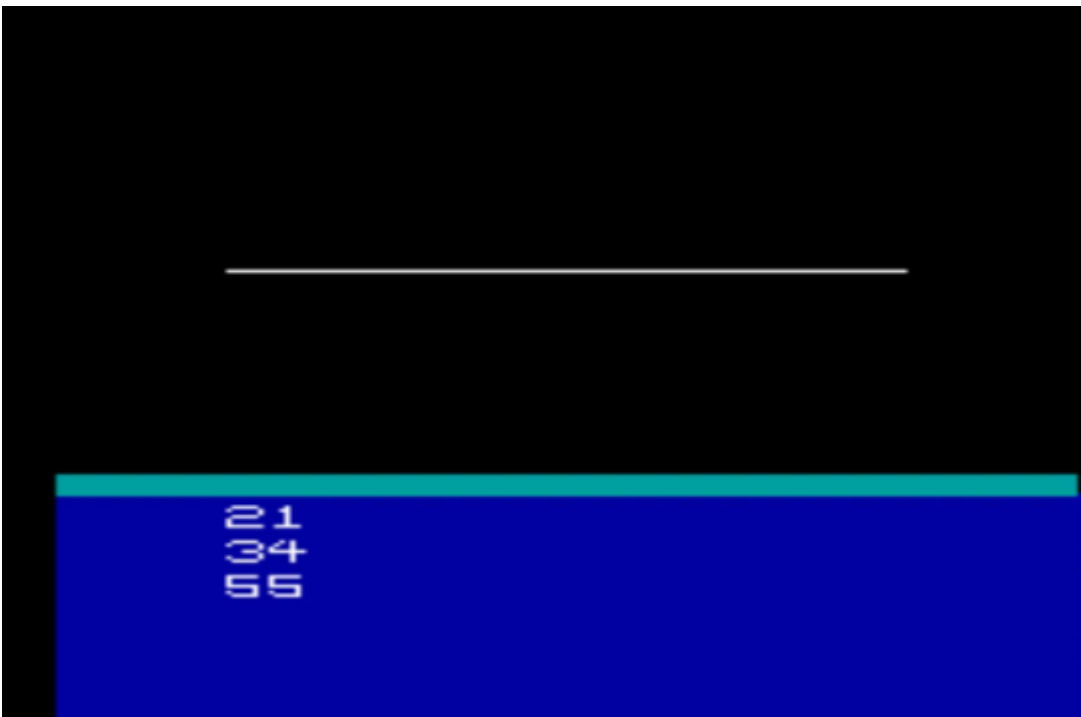
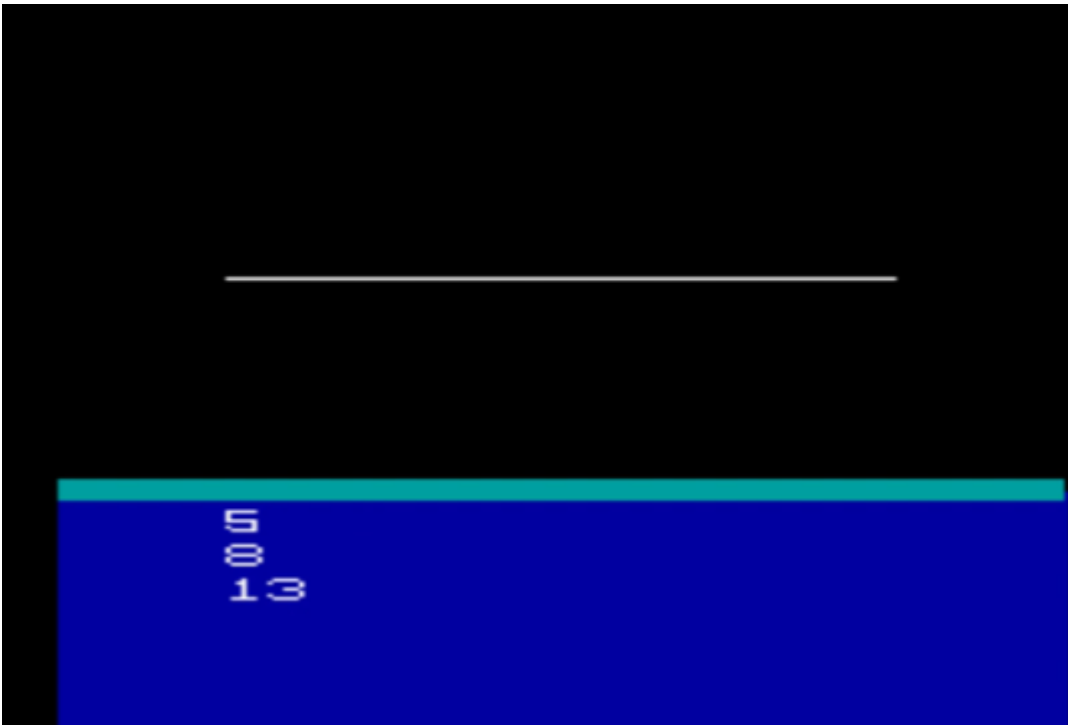
```
Y=Z
```

```
FIB
```

```
ENDP
```

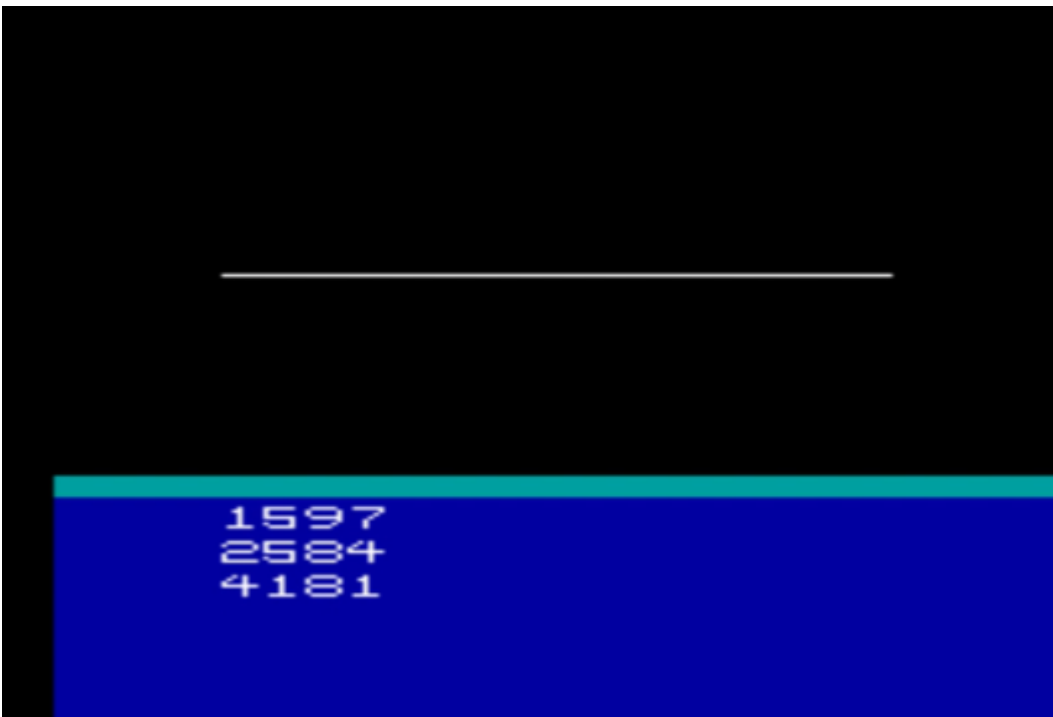
A short recursive program to list the sequence.





89
144
233

377
610
987



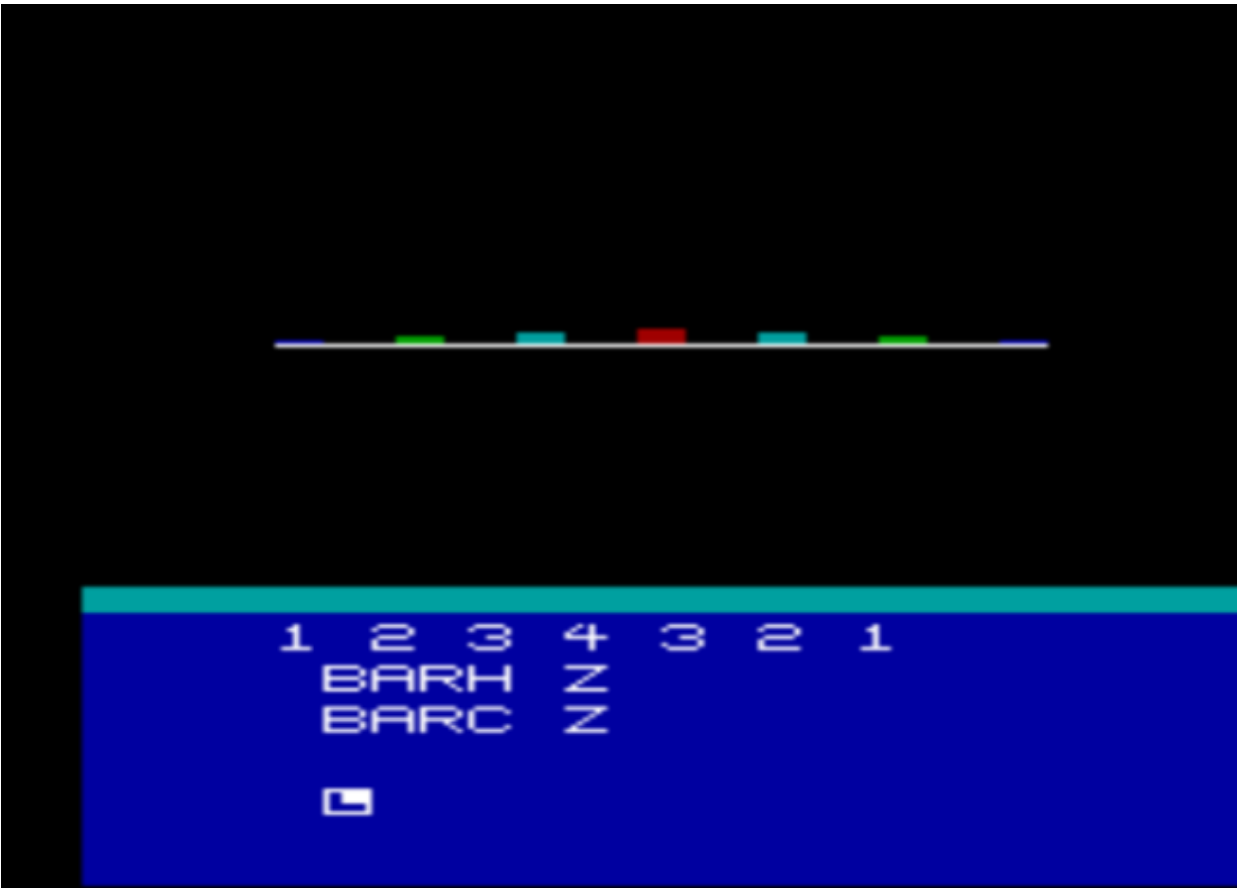
Stack overflow ends the program.

Pyramid chart

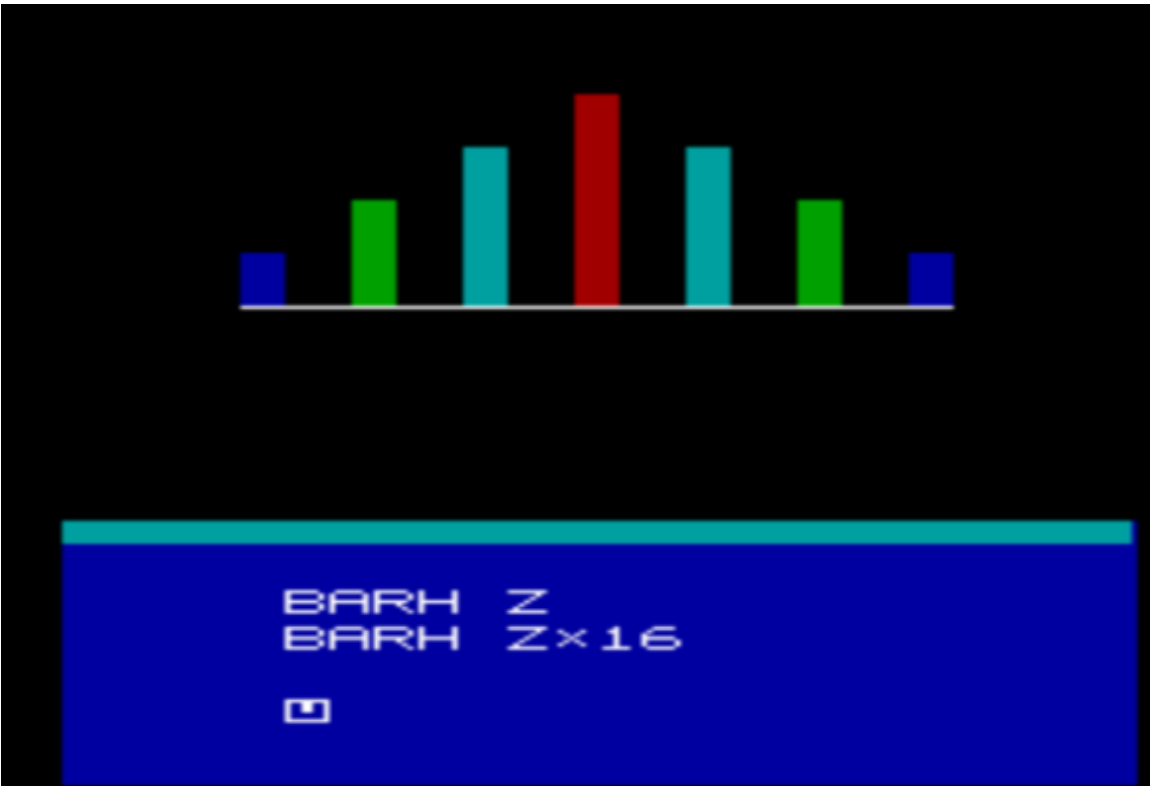


```
Z=Z MIN (8-Z)
Z
1 2 3 4 3 2 1
```

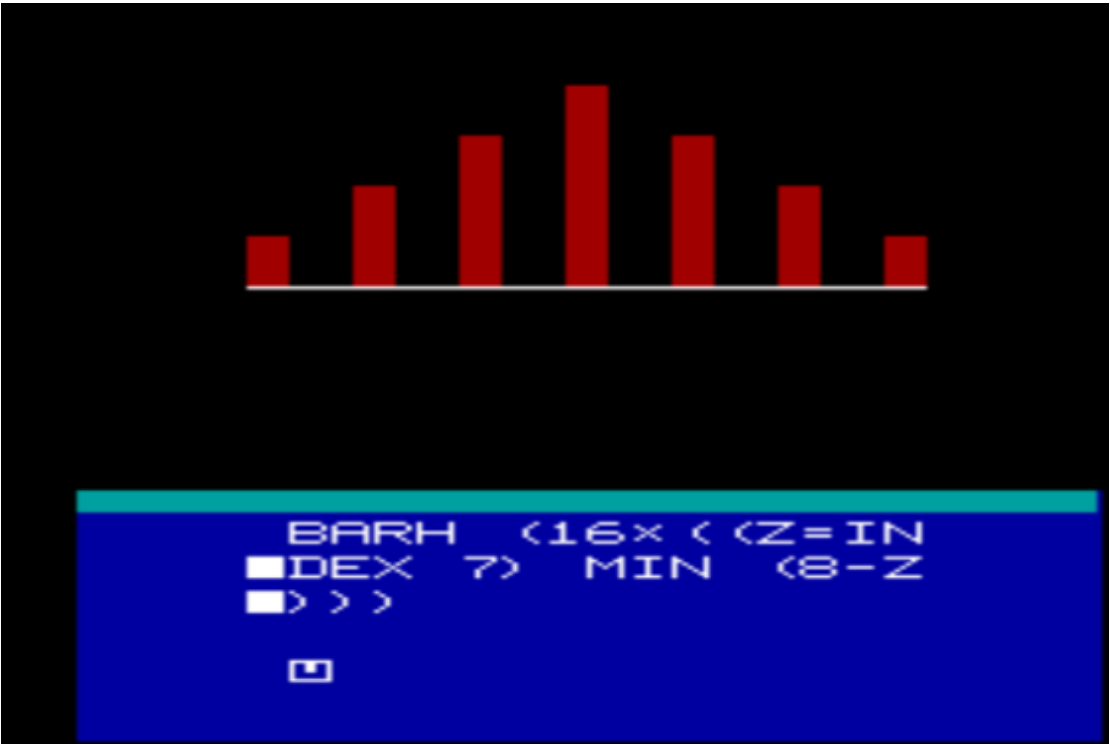




A small pyramid.



Scale it with multiplication.



As a one-liner.

Recursive Factorial Fun

In APL/S, recursive functions must account for all variables being global. Outside the program, set X and Y to 1, and assign the value whose factorial you want to N.

X=Y=1

N=10

PROG F

IF X LE N

THEN

Y=Y×X

X=X+1

F

ENDI

ENDP

X=Y=1
Z=1 0
F
Y
□

F
Y
36288 0 0
□

X=Y=1

N=6

PROG F

IF X LE N

THEN

Y=Y×X

X=X+1

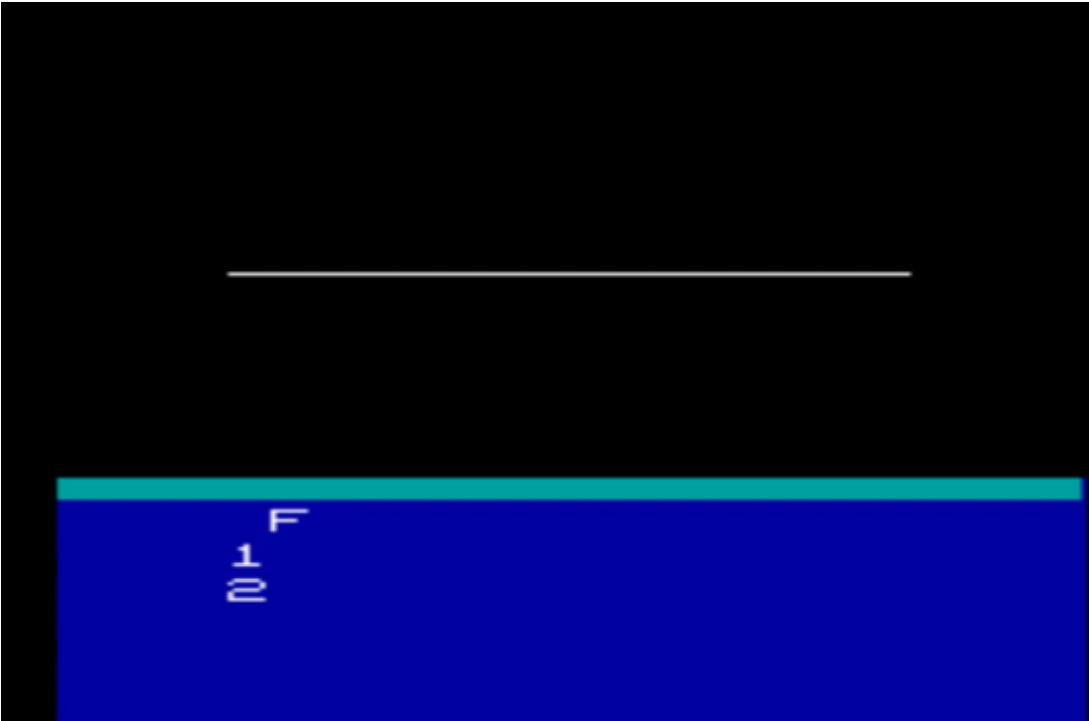
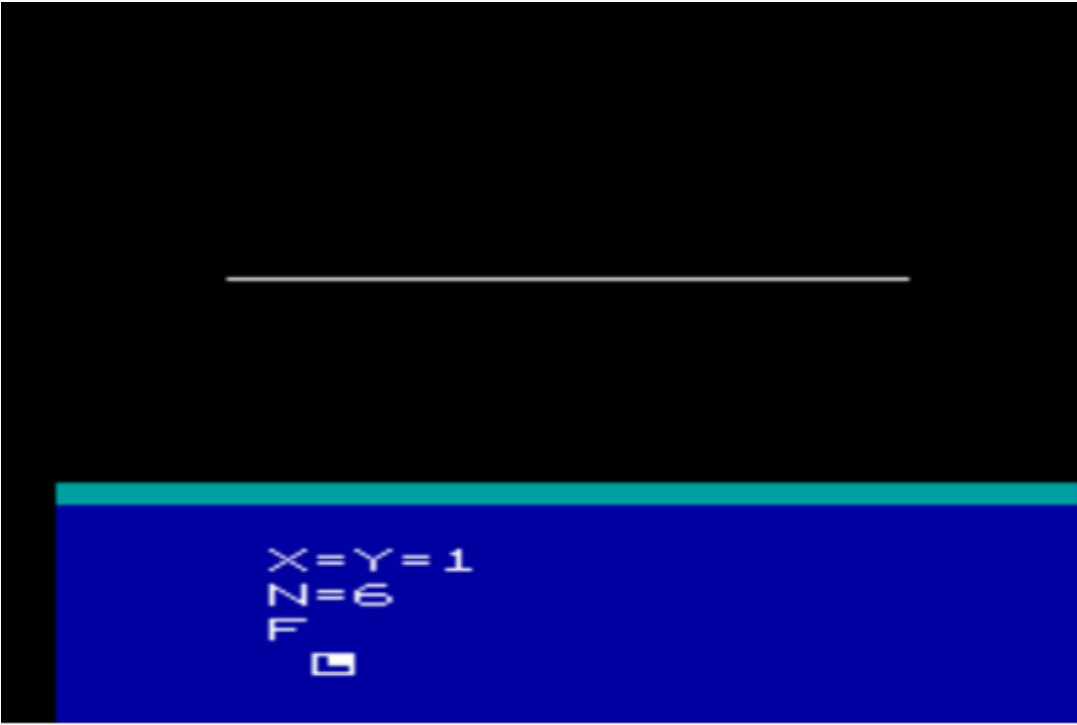
Y

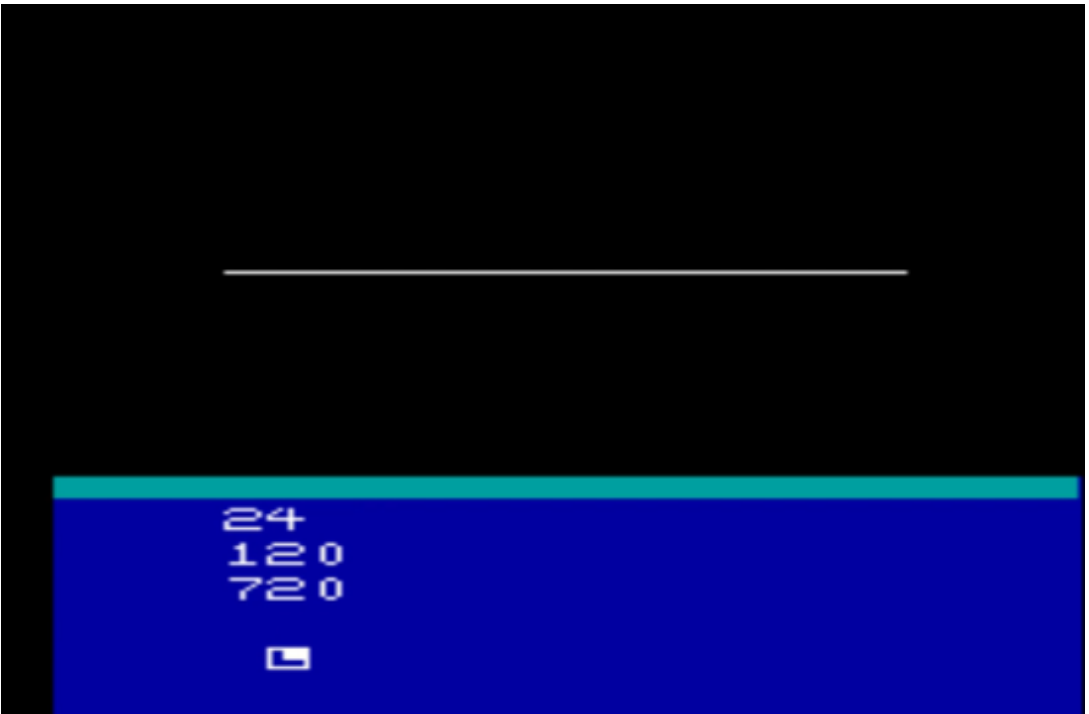
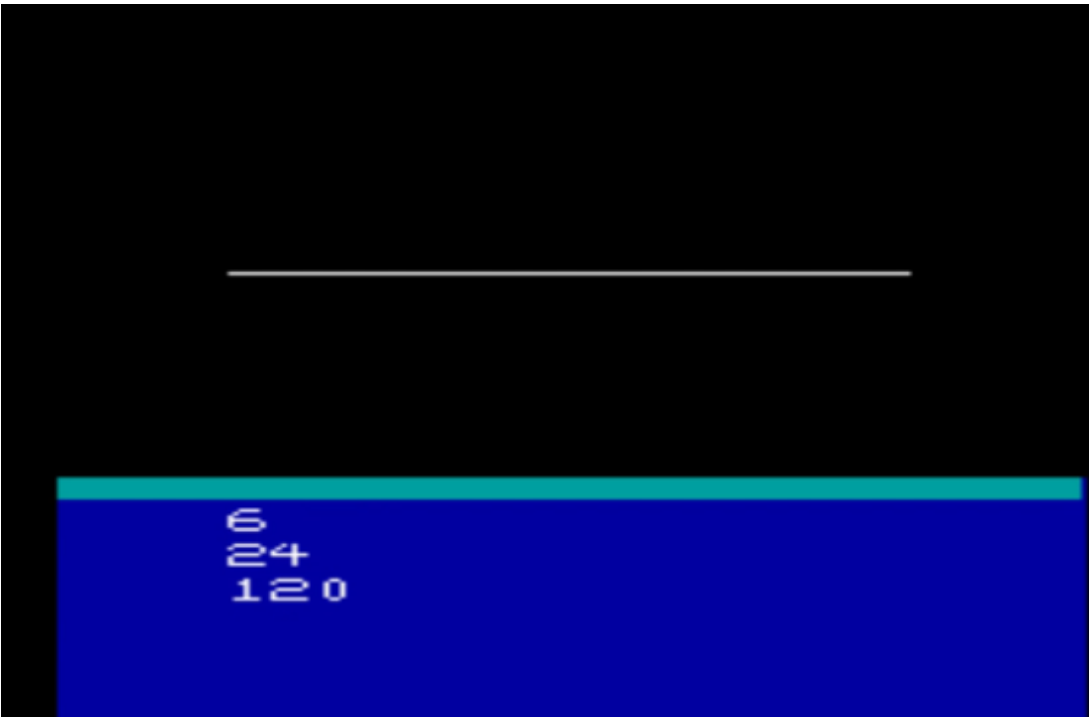
F

ENDI

ENDP

Add a line to print the interim result, Y.





```
720
X=Y=1
Z=31
F
□
```

8.84176E30
2.652528E32
8.222838E33



0.022838E33

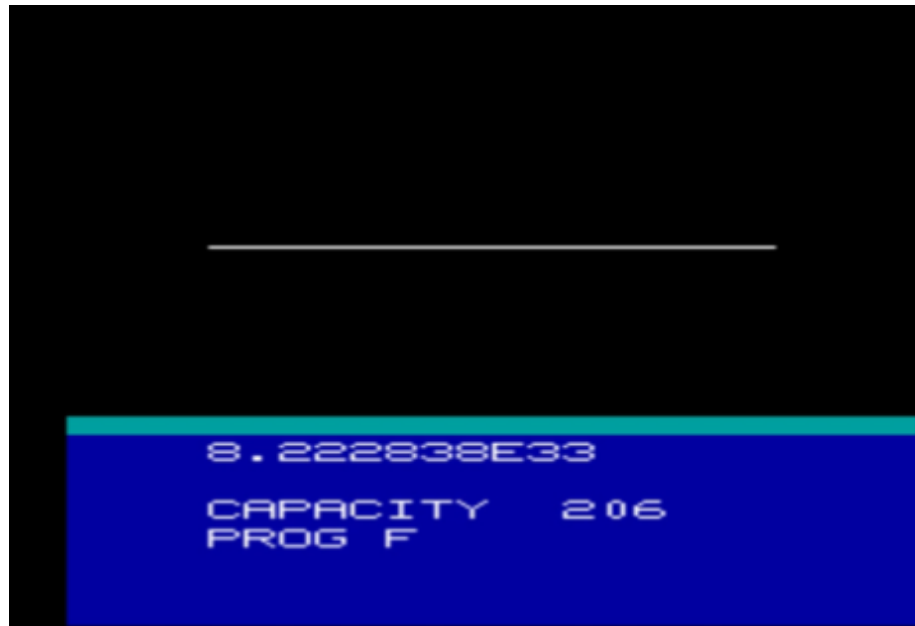
X=Y=1

Z=32

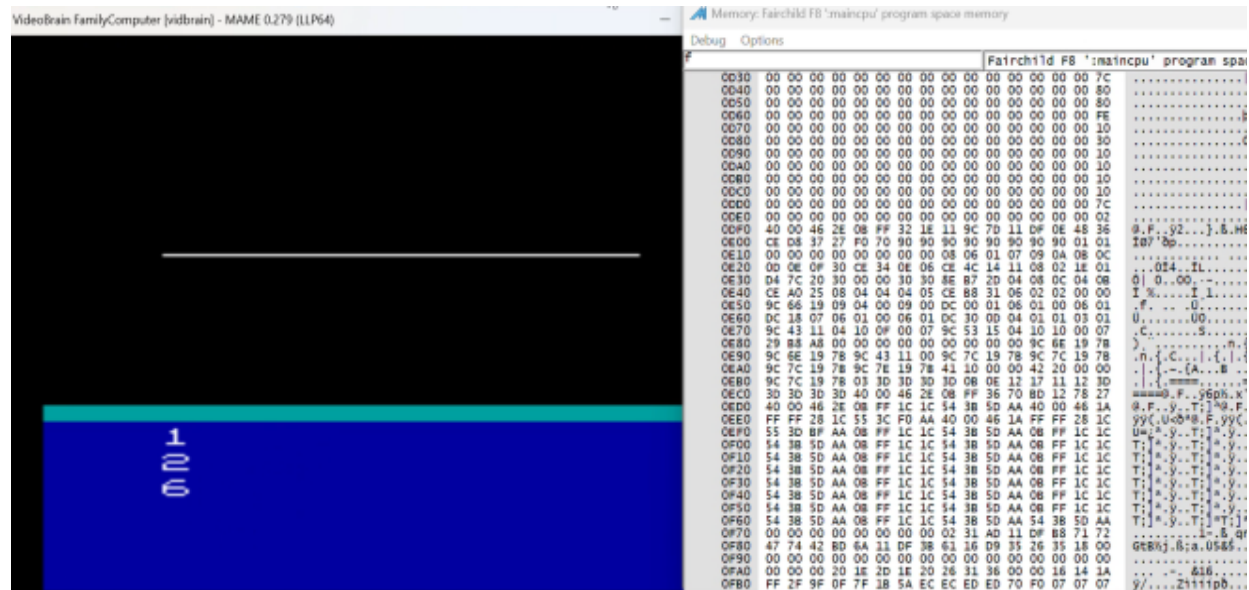
π

□

Capacity 206 is an “Out of stack space” error.



In the MAME debugger, you can see the stack grow as the program runs. Let's overflow the stack again while watching it fill up.

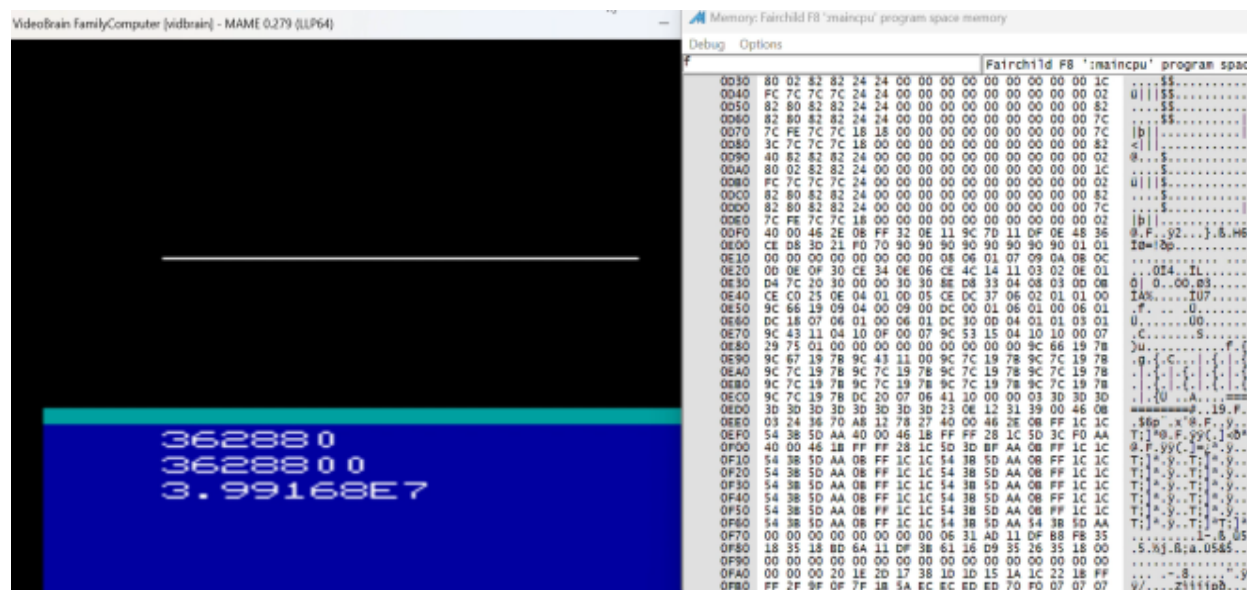


VideoBrain FamilyComputer [vidbrain] - MAME 0.279 (LLP64)

Memory: Fairchild F8 'maincpu' program space memory

Debug Options

Address	Hex Data	ASCII Data
0030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00F0	40 00 46 2E 08 FF 32 1E 11 9C 70 11 DF 0E 48 36	0.F.y2...].&.H6
0100	CE D8 37 27 F0 70 90 90 90 90 90 90 90 90 01 01	Id7'8p.....
0110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0120	00 0E 0F 30 CE 34 0E 06 CE 4C 14 11 08 02 1E 01	...014..IL.....
0130	D4 7C 20 30 00 00 30 30 8E 87 2D 04 08 0C 04 08	0 0..00.....
0140	CE A0 25 08 04 04 04 05 CE 88 31 06 02 02 00 00	I %....I 1....
0150	9C 66 19 09 04 00 09 00 DC 00 01 06 01 00 06 01	f.....U.....
0160	DC 18 07 06 01 00 06 01 DC 30 00 04 01 01 03 01	U.....00.....
0170	9C 43 11 04 10 0F 00 07 9C 53 15 04 10 10 00 07	.C.....S.....
0180	29 88 A8 00 00 00 00 00 00 00 00 00 00 00 00 00	>.....n..
0190	9C 66 19 78 9C 43 11 00 9C 7C 19 78 9C 7C 19 78	.n..C... .l..
01A0	9C 7C 19 78 9C 7C 19 78 41 10 00 00 42 20 00 00	(A...&..
01B0	9C 7C 19 78 03 30 3D 3D 3D 08 0E 12 37 11 12 3D	mmmm.....
01C0	3D 3D 3D 3D 40 00 46 2E 08 FF 36 70 8D 12 78 27	mmmm0.F..y6ph.x'
01D0	40 00 46 2E 08 FF 1C 1C 54 38 5D AA 40 00 46 1A	0.F..y..T..]0.F.
01E0	FF FF 28 1C 55 3C F0 AA 40 00 46 1A FF FF 28 1C	yyC(U0000.F.yyC.
01F0	55 3D BF AA 08 FF 1C 1C 54 38 5D AA 08 FF 1C 1C	T:'.y..T..y..
0200	54 38 5D AA 08 FF 1C 1C 54 38 5D AA 08 FF 1C 1C	T:'.y..T..y..
0210	54 38 5D AA 08 FF 1C 1C 54 38 5D AA 08 FF 1C 1C	T:'.y..T..y..
0220	54 38 5D AA 08 FF 1C 1C 54 38 5D AA 08 FF 1C 1C	T:'.y..T..y..
0230	54 38 5D AA 08 FF 1C 1C 54 38 5D AA 08 FF 1C 1C	T:'.y..T..y..
0240	54 38 5D AA 08 FF 1C 1C 54 38 5D AA 08 FF 1C 1C	T:'.y..T..y..
0250	54 38 5D AA 08 FF 1C 1C 54 38 5D AA 08 FF 1C 1C	T:'.y..T..y..
0260	54 38 5D AA 08 FF 1C 1C 54 38 5D AA 08 FF 1C 1C	T:'.y..T..y..
0270	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0280	47 74 42 8D 6A 11 DF 38 61 16 D9 35 16 35 18 00	ge8hj.R;a.0545...
0290	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
02A0	00 00 00 20 1E 2D 1E 2D 26 31 36 00 00 16 14 1A	...-.&16.....
02B0	FF 2F 9F 0F 7F 1B 5A EC EC ED ED 70 F0 07 07 07	y/....Z1111pb...

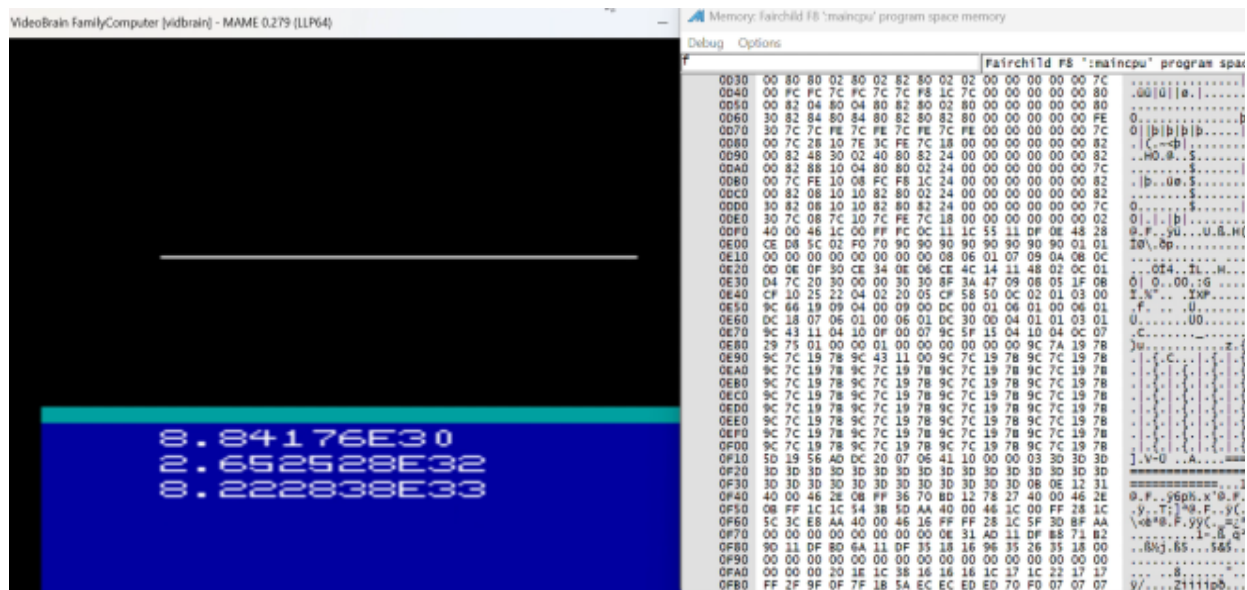


VideoBrain FamilyComputer [vidbrain] - MAME 0.279 (LLP64)

Memory: Fairchild F8 'maincpu' program space memory

Debug Options

Address	Hex Data	ASCII Data
0030	80 02 82 82 24 24 00 00 00 00 00 00 00 00 00 00	...\$......
0040	FC 7C 7C 7C 24 24 00 00 00 00 00 00 00 00 00 00	U \$......
0050	82 80 82 82 24 24 00 00 00 00 00 00 00 00 00 00	...\$......
0060	82 80 82 82 24 24 00 00 00 00 00 00 00 00 00 00	...\$......
0070	7C FE 7C 7C 18 00 00 00 00 00 00 00 00 00 00 00	b
0080	3C 7C 7C 7C 18 00 00 00 00 00 00 00 00 00 00 00	<
0090	40 82 82 82 24 24 00 00 00 00 00 00 00 00 00 00	0...\$......
00A0	80 02 82 82 24 24 00 00 00 00 00 00 00 00 00 00	...\$......
00B0	FC 7C 7C 7C 24 24 00 00 00 00 00 00 00 00 00 00	U \$......
00C0	82 80 82 82 24 24 00 00 00 00 00 00 00 00 00 00	...\$......
00D0	82 80 82 82 24 24 00 00 00 00 00 00 00 00 00 00	...\$......
00E0	7C FE 7C 7C 18 00 00 00 00 00 00 00 00 00 00 00	b
00F0	40 00 46 2E 08 FF 32 0E 11 9C 70 11 DF 0E 48 36	0.F.y2...].&.H6
0100	CE D8 3D 21 F0 70 90 90 90 90 90 90 90 90 01 01	Id7'8p.....
0110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0120	00 0E 0F 30 CE 34 0E 06 CE 4C 14 11 03 02 0E 01	...014..IL.....
0130	D4 7C 20 30 00 00 30 30 8E D8 33 04 08 03 00 08	0 0..00.03....
0140	CE C0 25 0E 04 01 00 05 CE DC 37 06 02 01 01 00	IAn...IU7.....
0150	9C 66 19 09 04 00 09 00 DC 00 01 06 01 00 06 01	f.....U.....
0160	DC 18 07 06 01 00 06 01 DC 30 00 04 01 01 03 01	U.....00.....
0170	9C 43 11 04 10 0F 00 07 9C 53 15 04 10 10 00 07	.C.....S.....
0180	29 75 01 00 00 00 00 00 00 00 00 00 00 00 00 00	>.....n..
0190	9C 67 19 78 9C 43 11 00 9C 7C 19 78 9C 7C 19 78	.n..C... .l..
01A0	9C 7C 19 78 9C 7C 19 78 9C 7C 19 78 9C 7C 19 78	(A...&..
01B0	9C 7C 19 78 03 30 3D 3D 3D 0E 12 31 39 00 46 08	(U...A...mm
01C0	3D 3D 3D 3D 40 00 46 2E 08 FF 36 70 8D 12 78 27	mmmmmm0.F..y6ph.x'
01D0	03 24 36 70 A8 11 78 27 40 00 46 2E 08 FF 1C 1C	T:'.y..T..]0.F.
01E0	54 38 5D AA 40 00 46 18 FF FF 28 1C 5D 3D BF AA 08	T:'.y..T..]0.F.yyC..]0.F.
01F0	40 00 46 18 FF FF 28 1C 5D 3D BF AA 08 FF 1C 1C	T:'.y..T..y..
0200	54 38 5D AA 08 FF 1C 1C 54 38 5D AA 08 FF 1C 1C	T:'.y..T..y..
0210	54 38 5D AA 08 FF 1C 1C 54 38 5D AA 08 FF 1C 1C	T:'.y..T..y..
0220	54 38 5D AA 08 FF 1C 1C 54 38 5D AA 08 FF 1C 1C	T:'.y..T..y..
0230	54 38 5D AA 08 FF 1C 1C 54 38 5D AA 08 FF 1C 1C	T:'.y..T..y..
0240	54 38 5D AA 08 FF 1C 1C 54 38 5D AA 08 FF 1C 1C	T:'.y..T..y..
0250	54 38 5D AA 08 FF 1C 1C 54 38 5D AA 08 FF 1C 1C	T:'.y..T..y..
0260	54 38 5D AA 08 FF 1C 1C 54 38 5D AA 08 FF 1C 1C	T:'.y..T..y..
0270	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0280	18 35 18 BD 6A 11 DF 38 61 16 D9 35 16 35 18 00	..S.hj.R;a.0545...
0290	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
02A0	00 00 00 20 1E 2D 17 38 1D 1D 15 1A 1C 22 18 FF	...-.&16.....
02B0	FF 2F 9F 0F 7F 1B 5A EC EC ED ED 70 F0 07 07 07	y/....Z1111pb...

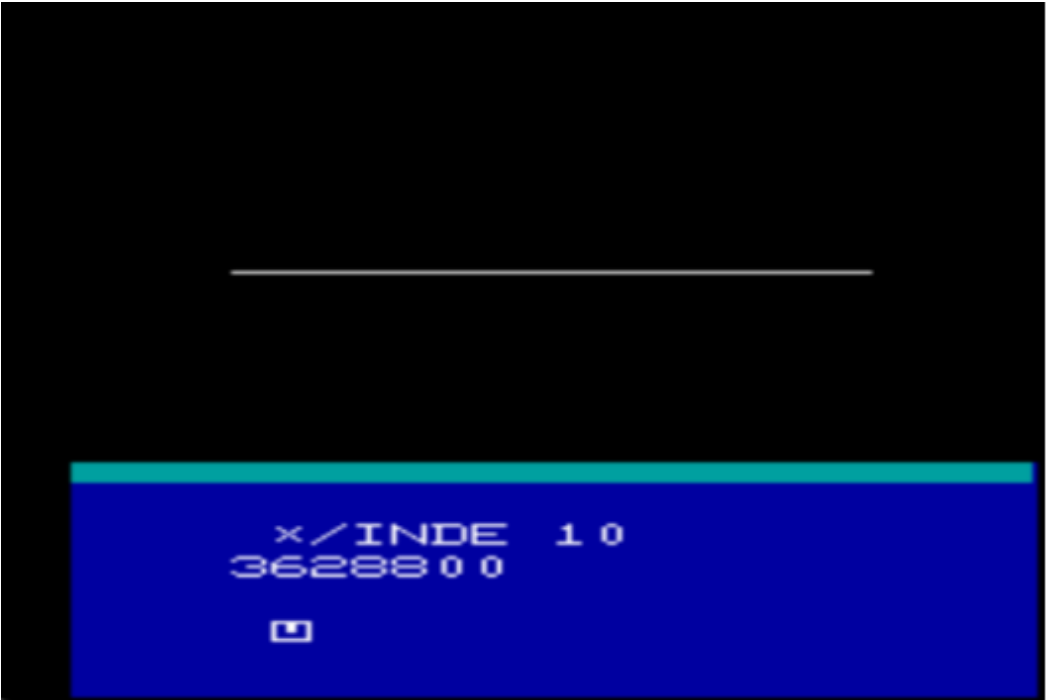


55
1.26964E73

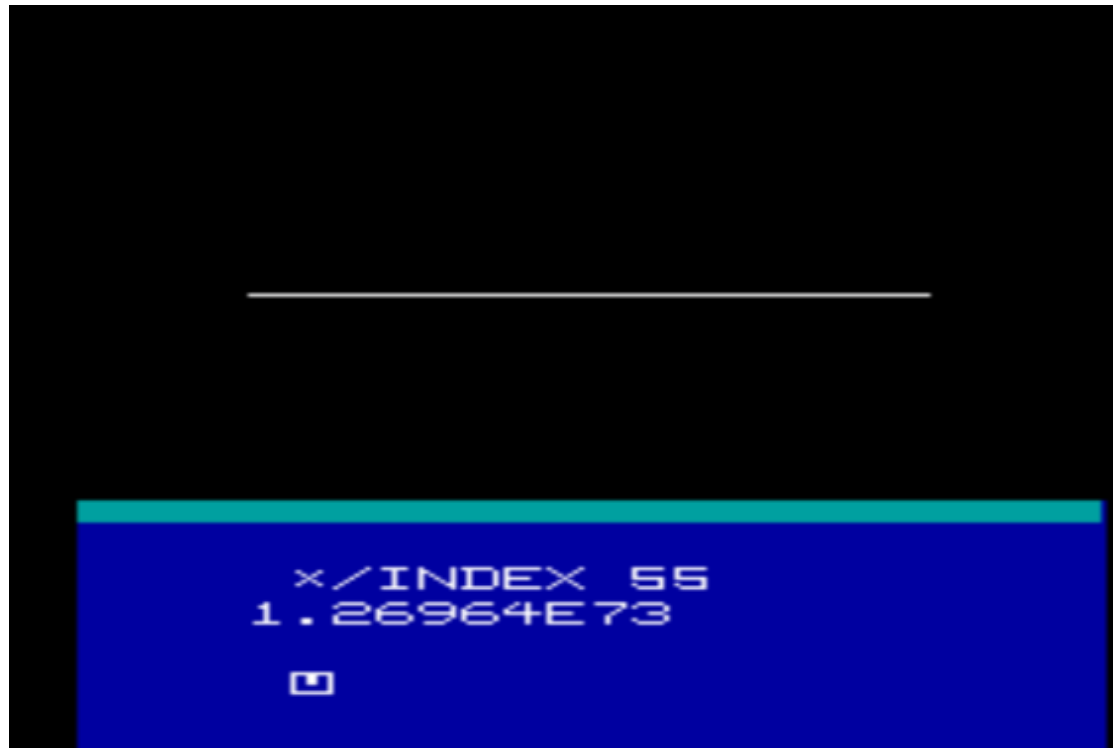




Barheight accepts values over 64 but cuts them off.



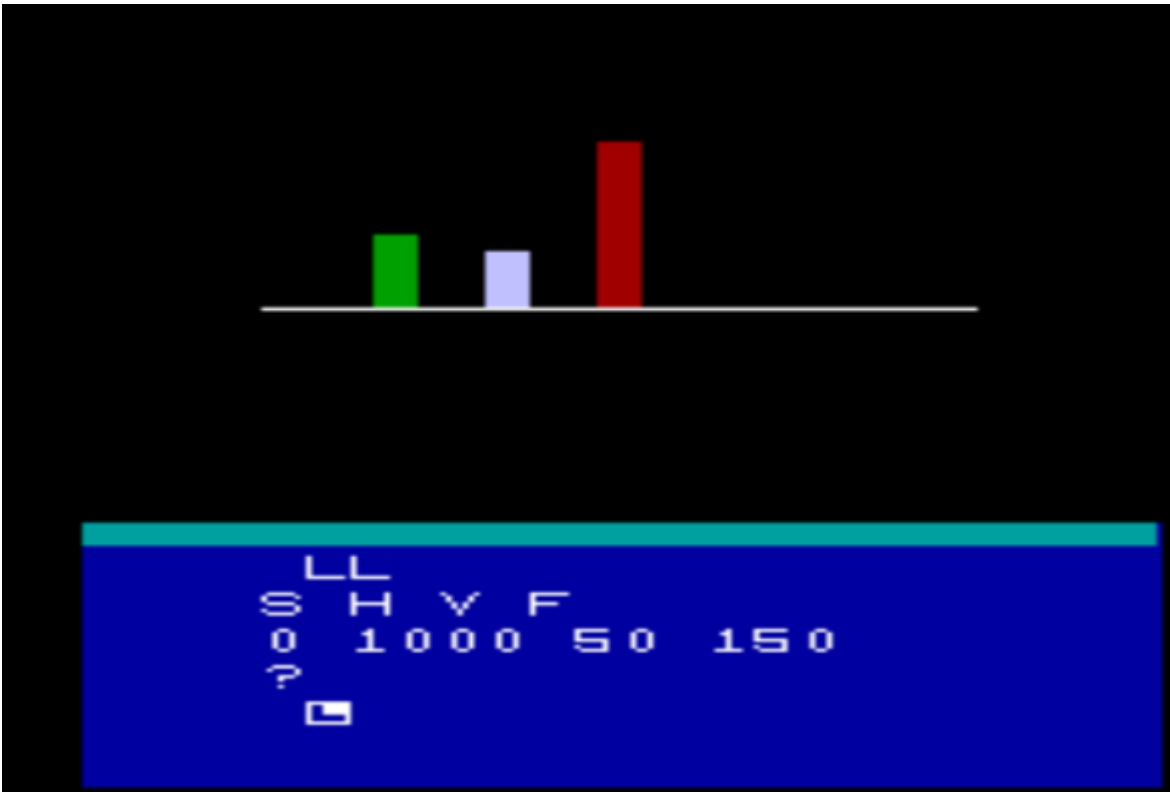
Times reduction can also be used to compute factorials.



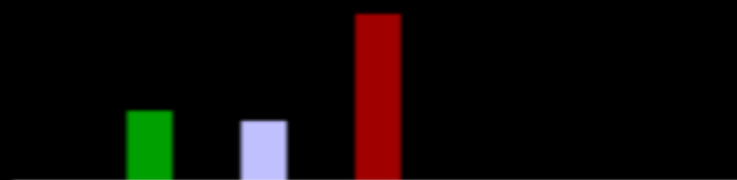
Lunar Lander

This program is a port of David Ahl's 1973 BASIC game [ROCKT1](#). Be sure to check the APL/S product brochure for another [version](#) of Lunar Lander.

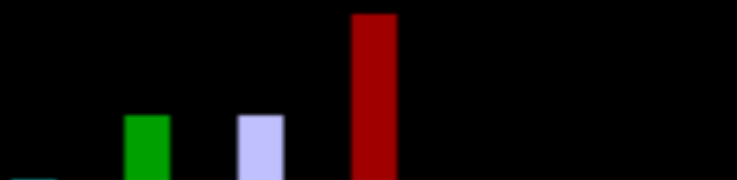
```
PROG LL
T=0
H=1000
V=50
F=150
BARC 3 2 9 4
WHILE H GT 0
DO
  "S H V F"
  T,H,V,F
  BARH T,(H+15),V,F+3
  B=KEYB
  IF B GT F
  THEN B=F
  ENDI
  V1=V-B+5
  F=F-B
  H=H-.5x(V+V1)
  IF H GT 0
  THEN
    T=T+1
    V=V1
    IF F LE 0
    THEN "OUT OF FUEL"
      WHILE H GT 0
      DO
        T,H,V,F
        BARH T,(H+15),V,F+3
        V1=V+5
        H=H-.5x(V+V1)
        T=T+1
        V=V1
      ENDW
    ENDI
  ELSE EXIT
  ENDI
ENDW
"S H V F"
T,H,V,F
"CONTACT"
ENDP
```



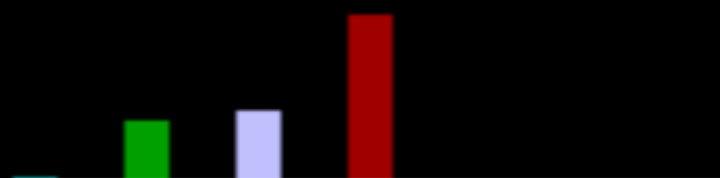
A sample run.



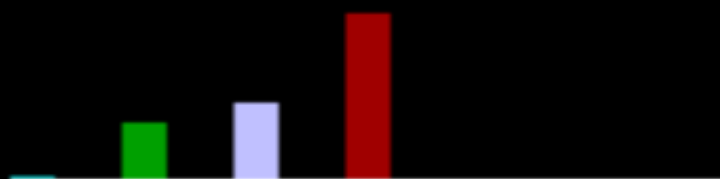
ρ_1
 S H V F
 1 948 54 149
 ρ_0
 0



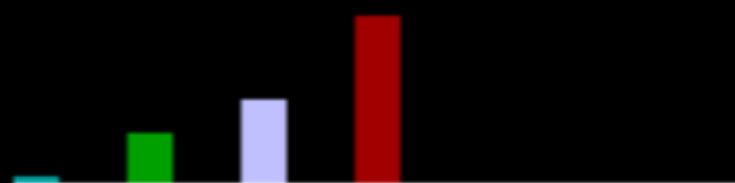
ρ_0
 000 H V F
 000 891.5 59 149
 ρ_0
 0



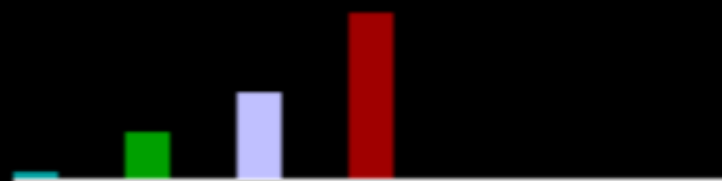
ρ_0
 S H V F
 3 830 64 149
 ρ_0
 9



ρ_0
 S H V F
 4 763.5 69 149
 ρ_0
 9

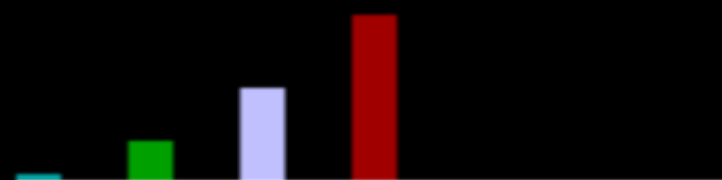


```
0
55 H V F
0 692 74 149
0
```



```

? 0
S  H  V  F
6  615.5  79  149
? 0
  
```



```

? 0
S  H  V  F
7  534  84  149
? 44
  
```



```

?44
S H V F
8 469.5 45 105
?0
  
```



```

?0
S H V F
10 369.5 55 105
?
  
```




```

? 0
S  H  V  F
11  312  60  105
?
  
```



```

? 0
S  H  V  F
12  249.5  65  105
?
  
```



```

?22
S H V F
13 193 48 83
?
  
```



```

?0
S H V F
14 142.5 53 83
?
  
```



? 0
S H V F
15 87 58 83
?
□



? 33
S H V F
16 43 30 50
?
□



?5
S H V F
17 13 30 45
?
□



S H V F
17 13 30 12
CONTACT
□

Unit Circle

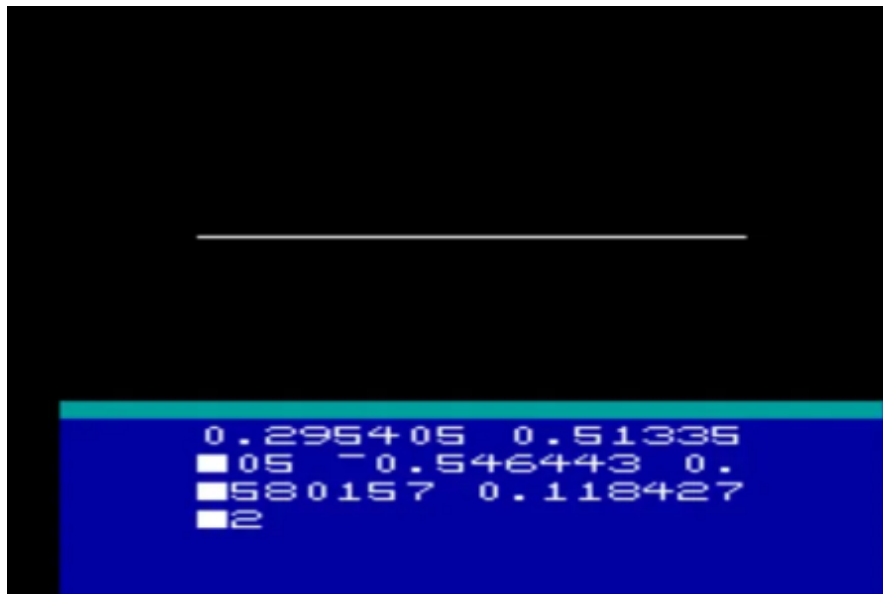
In his September, 1980 ACM Quote Quad paper [“Uniform sampling on simplices and spheres,”](#) mathematician Balder Von Hohenbaiken details his APL implementation of an algorithm presented by [Donald Knuth in *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*](#). Knuth shows how to use the polar method for generating normalized random deviates, developed by [Box, Muller, and Marsagli](#). We will use Von Hohenbaiken’s technique to generate and display unit circles using APL/S.

```
G=ARRAY 7

PROG UNI
BARC INDEX 7
X=INDEX 0
D=2
WHILE D GT SIZE X
DO
  WHILE 1
  DO
    V= -1+2*31
    V= -1+((RAND(2×V,(2×V)))÷V)
    IF 1 GT (S÷/(V*2))
    THEN
      EXIT
    ENDI
  ENDW
  X=X,V×(((−2×LOG S)÷S)*.5)
ENDW
IF 1 EQ (D MOD 2) // odd dimensions 3, 5, 7 etcetera
THEN
  M=ARRAY D
  MI=1
  WHILE MI LE D
  DO
    M(MI) = X(MI)
    MI=MI+1
  ENDW
  X=M
ENDI
X=X÷(÷/(X*2)*.5)
X
IF D EQ 2
THEN
  I=CEIL (3.5+(X(1)×3))
  H=CEIL (X(2)×64)
  G(I) = H
  BARH G
ENDI
ENDP
```

In APL/S, all variables are globals. Before running the program, first prepare a global array to store the bar graph height values. As the program is run multiple times, the bar graph will accumulate coordinates.

The [Box-Muller-Marsaglia-Knuth](#) method can generate coordinates for any dimension, although for our unit circle visualization we will only use $D=2$. This APL/S implementation can compute coordinates for spheres of higher dimension. Let's try an example run with $D=5$:



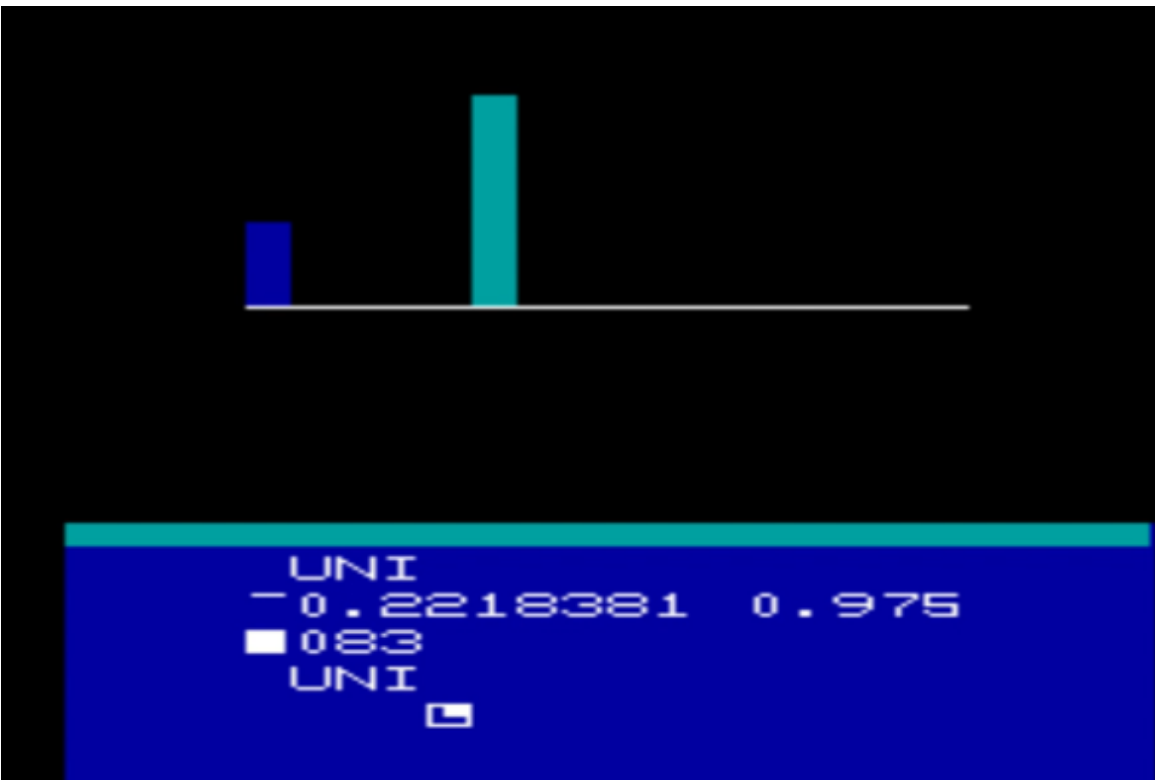
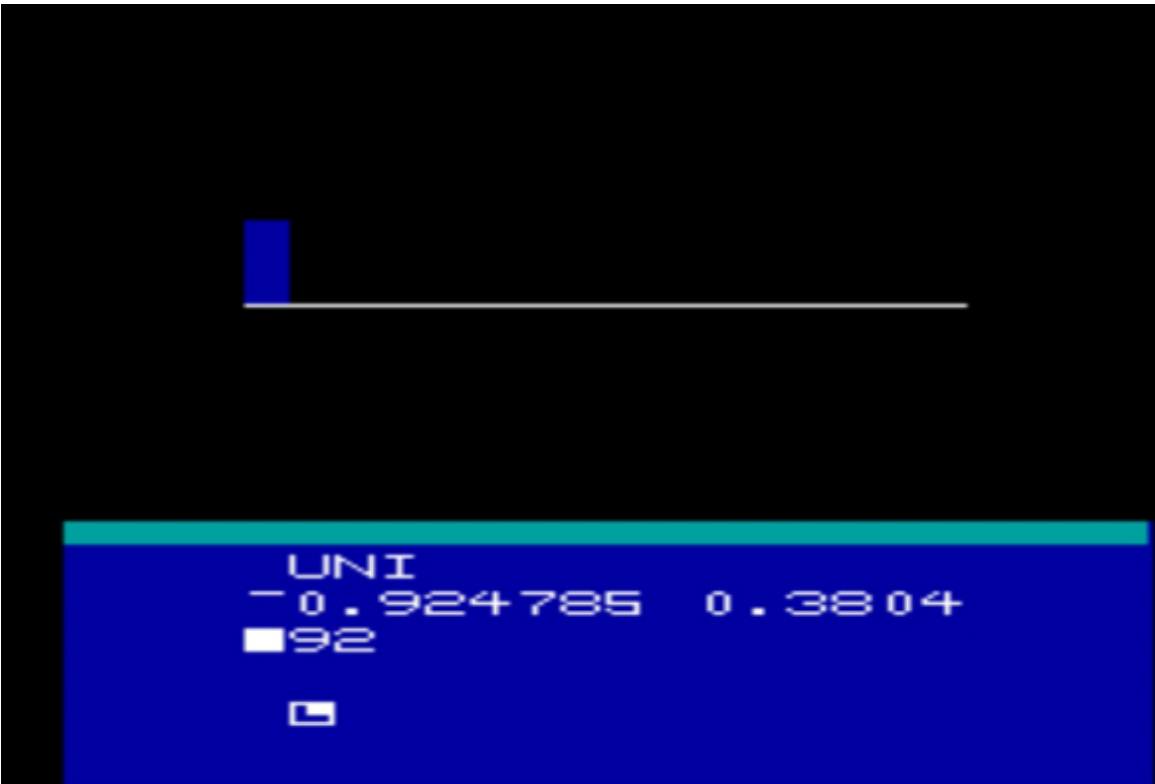
It's easy to check these coordinates using a calculator:

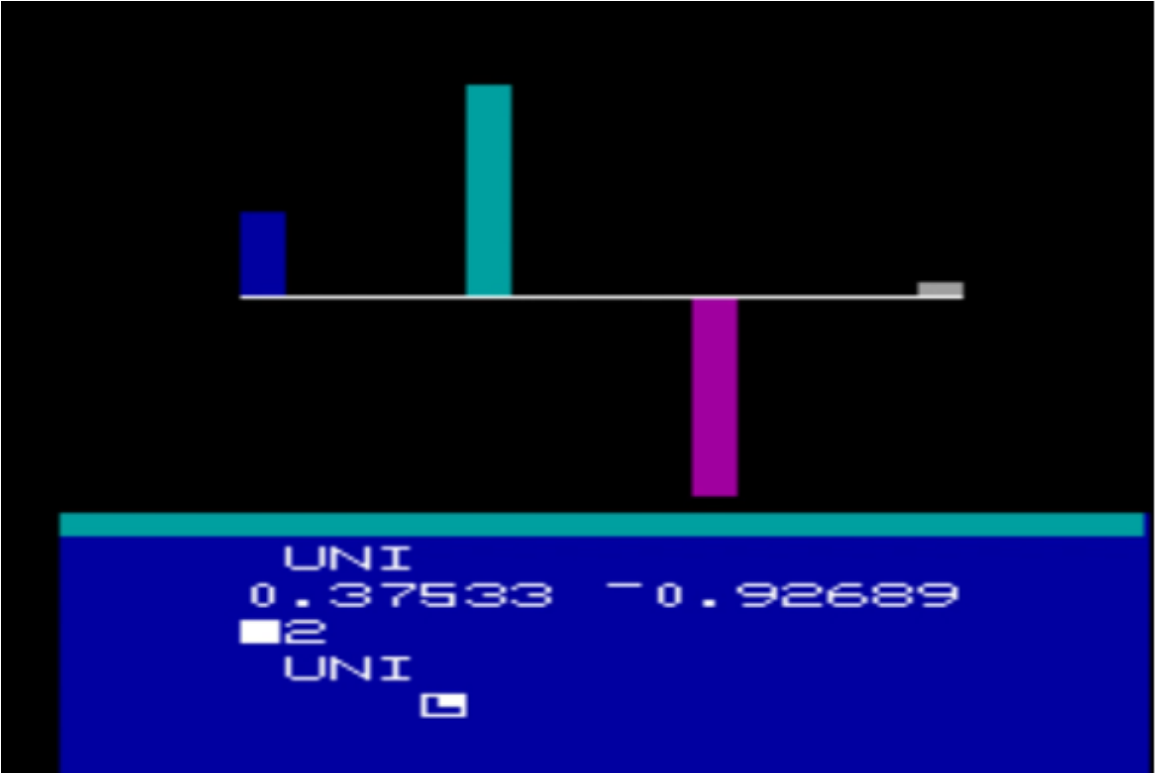
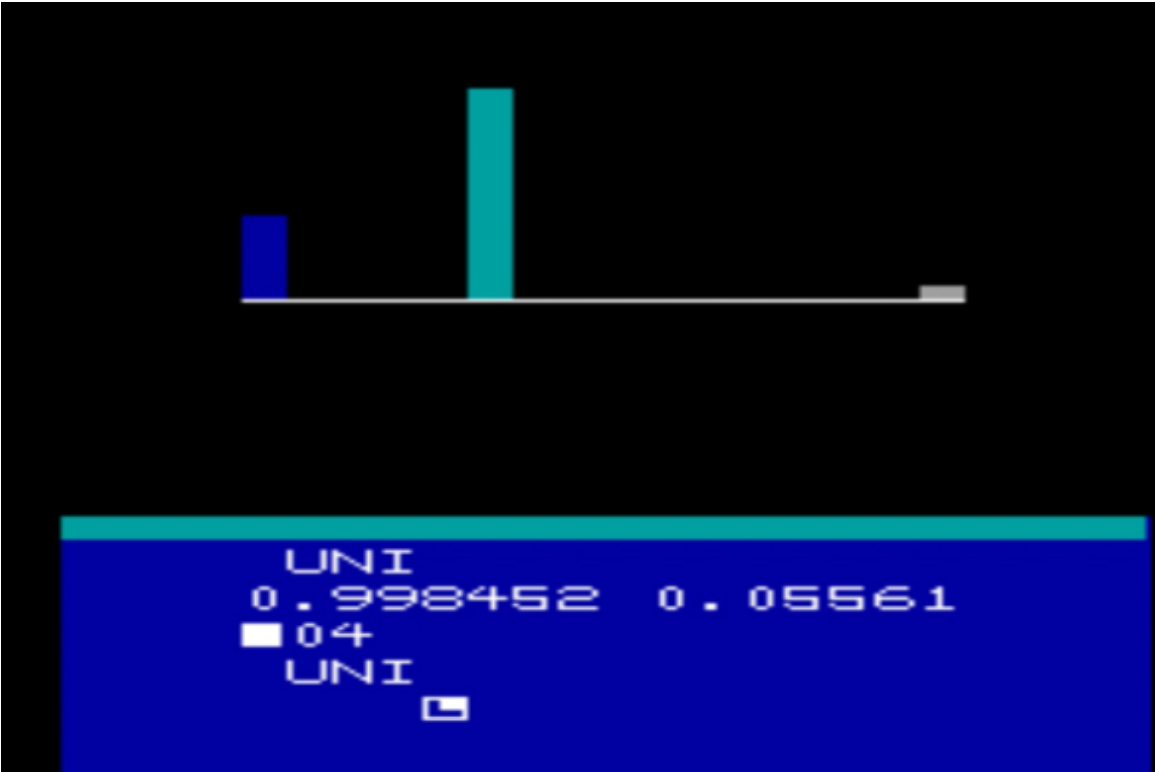
$\text{sqr}(0.2954) + \text{sqr}(0.51335) + \text{sqr}(-0.5464) + \text{sqr}(0.58015) + \text{sqr}(0.1184) = >$

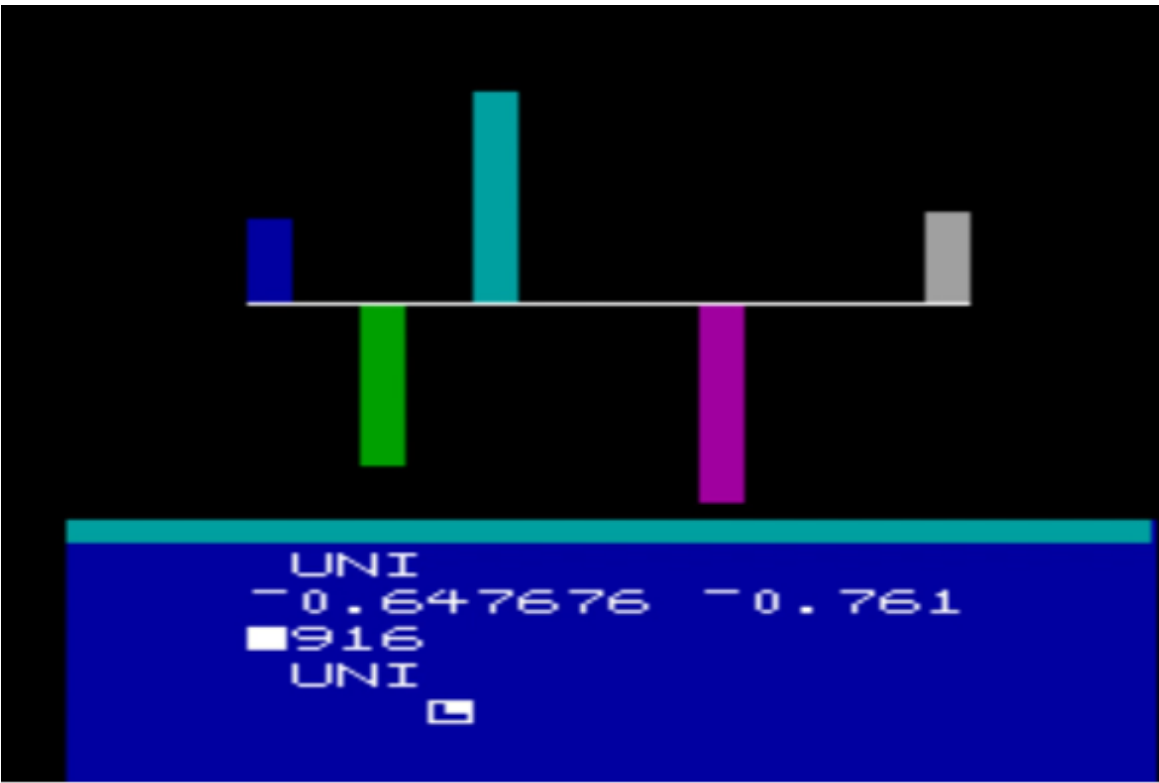
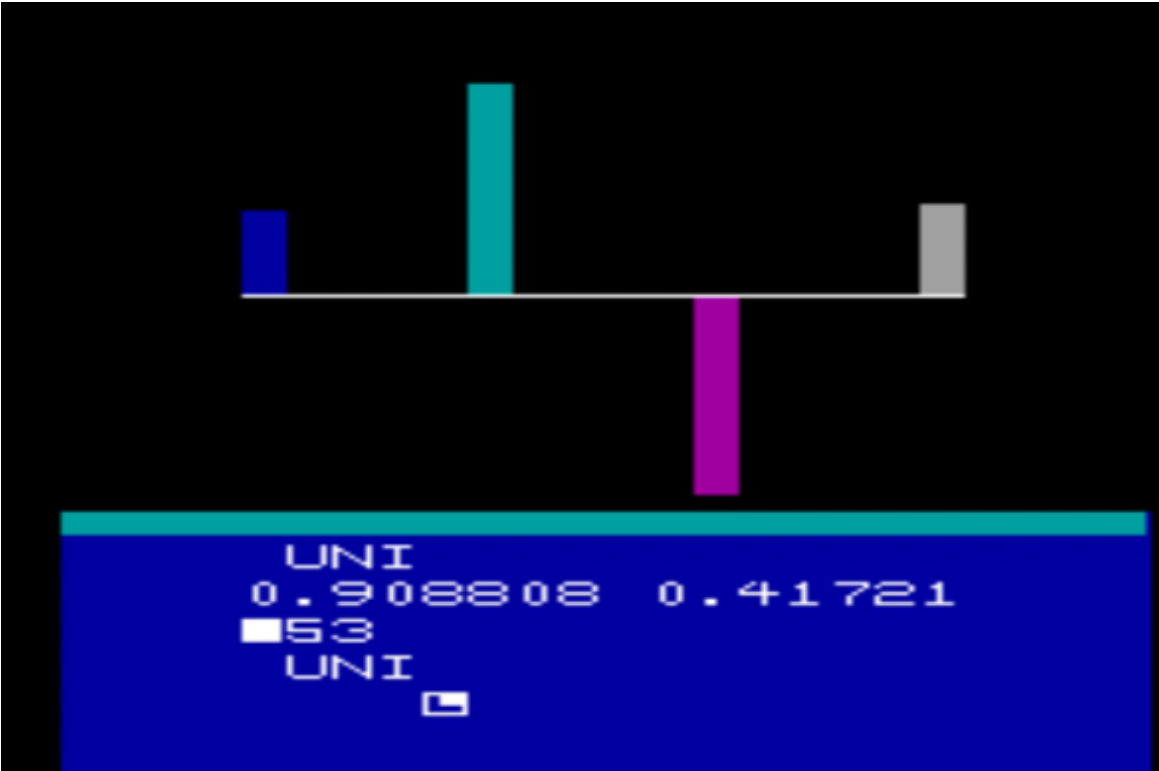
0.999934925

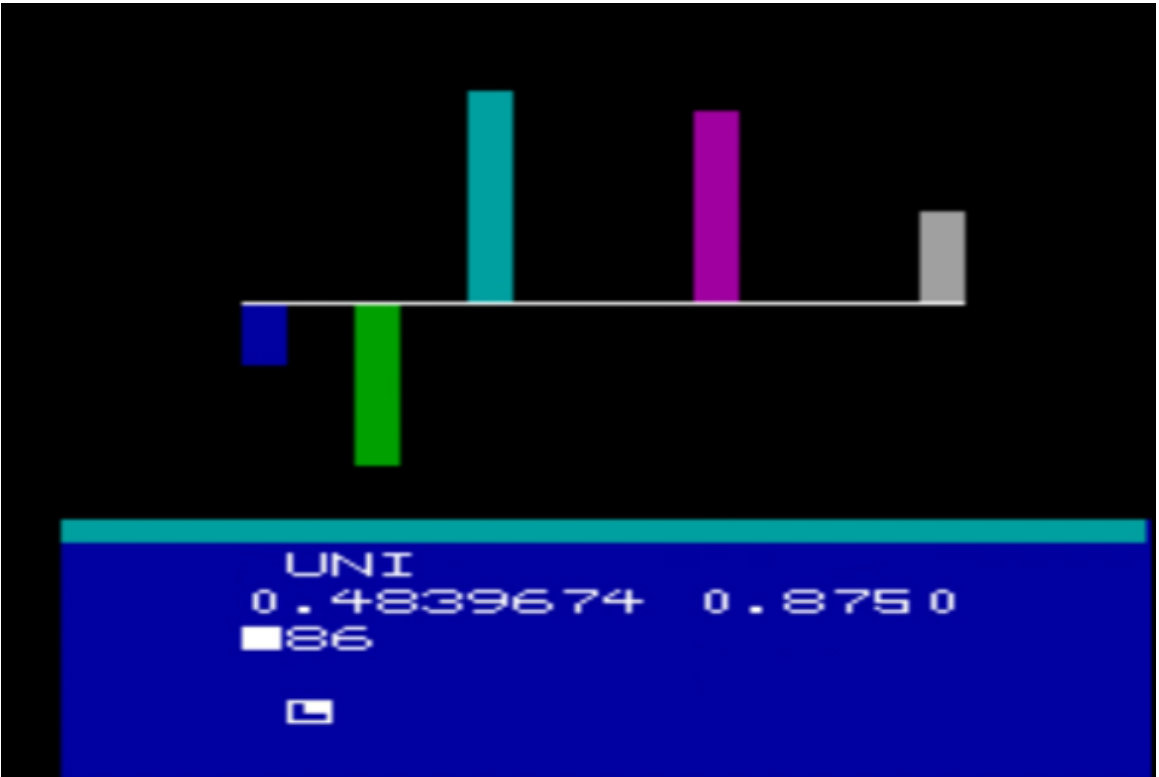
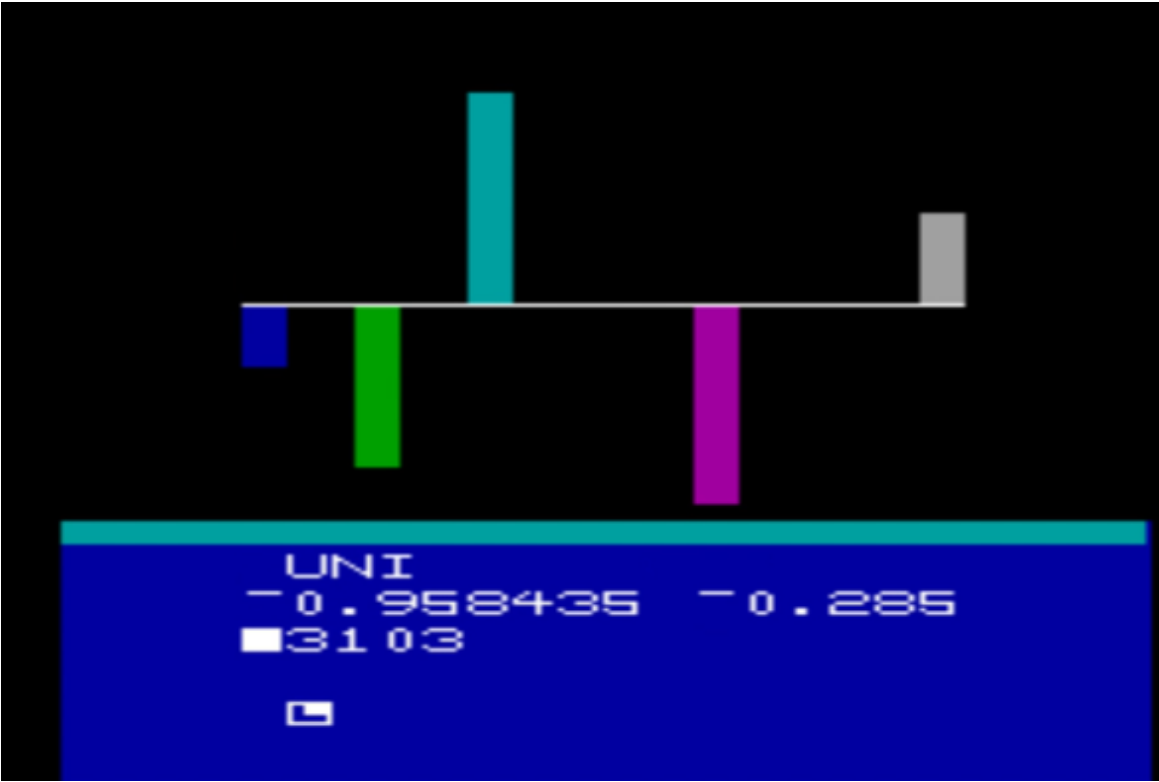
For odd-numbered dimensions, special handling is required since the algorithm generates $X[1]$, $X[2]$ pairs. In the odd cases of $D=3$, $D=5$, $D=7$, ... the line “IF 1 EQ (D MOD 2)” evaluates as true and then, somewhat awkwardly, a new array is created and all the pairs are copied into the new array except for the last pair, where only one member of the pair comes over. This newly created array has “dropped” half a pair, leaving an odd-numbered count of random deviates, done this way because there is no Drop in APL/S. After this, the norm of the resulting vector is computed as usual with $X=X \div (+/(X*2)*.5)$.

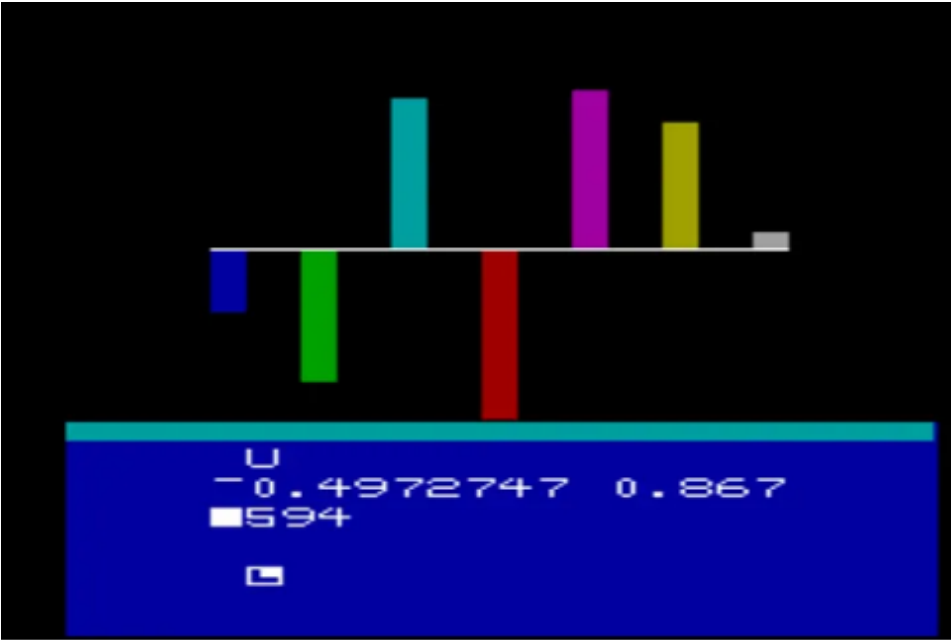
Finally, the program checks if $D=2$, and if so, scales and shifts the unit circle coordinates to ready them for bar graphing. The VideoBrain’s [bar graph facility offers 7 bars, with each bar capable of displaying values from -64 to +64](#). We can directly visualize the unit circle using this facility:



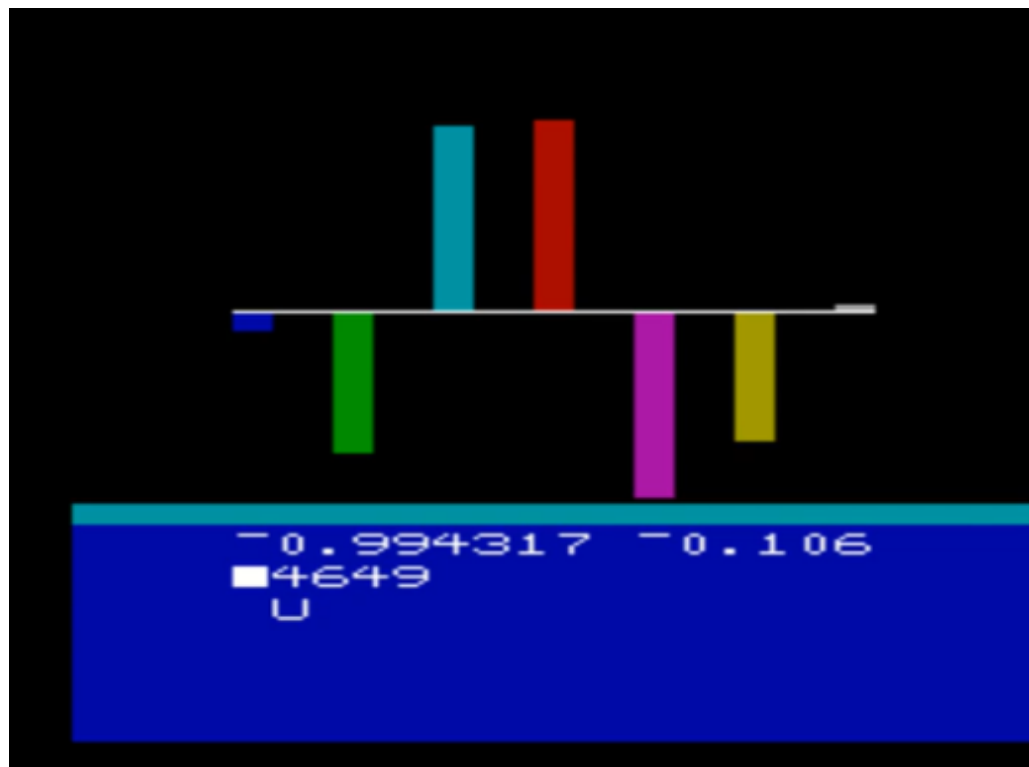


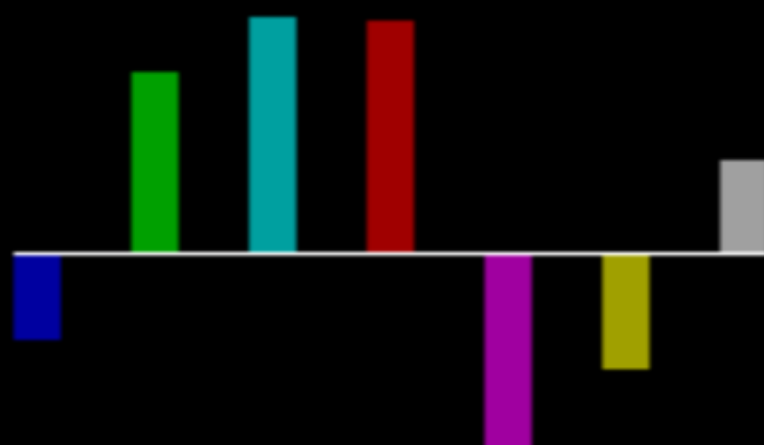






Another example of showing points being added to a unit circle chart.





■ 141
-0.870031 -0.492
■ 9972
□

For more, including Commodore 64, C128, and SuperPET implementations,
see “[Using APL to Visualize the Unit Circle on Vintage 8-bit Computers](#)”

Dicejack

A dice version of Blackjack, similar to the game “[Twenty-One](#)” from “[The Gateway Guide To The ZX81 And ZX80](#)” by [Mark Charlton](#). The book is part of the Creative Computing material released into the public domain by David Ahl.

```
PROG D
BARC 2 3
BARH +/(Z=Rand 6 6),0
Z
WHIL 1
DO
  'HIT? 1/0'
  C=KEYB
  IF C EQ 0
  THEN EXIT
  ENDI
  IF C EQ 1
  THEN Z=Z,RAND 6
  Z
  BARH +/Z
  IF +/Z GT 21
  THEN "BUSTED"
  EXIT
  ENDI
ENDI
ENDW
+/Z
IF +/Z LE 21
THEN
  B=0 //DEALER NOT BUSTED
  X=RAND 6 6
  X
  WHIL (+/Z) GE (+/X)
  DO
    X=X,RAND 6
    X
    BARH +/Z,+/X
    IF +/X GT 21
    THEN "I BUSTED"
    B=1
    EXIT
  ENDI
ENDW
+/X
IF B EQ 1
THEN "YOU WIN"
ELSE
  IF +/X GT +/Z
  THEN "I WIN"
  ELSE "YOU WIN"
  ENDI
ENDI
ENDI
ENDP
```



D
3 3
HIT? 1/0
?1
□



?1
3 3 3
HIT? 1/0
?1
□



?1
3 3 3 4
HIT? 1/0
?1
□



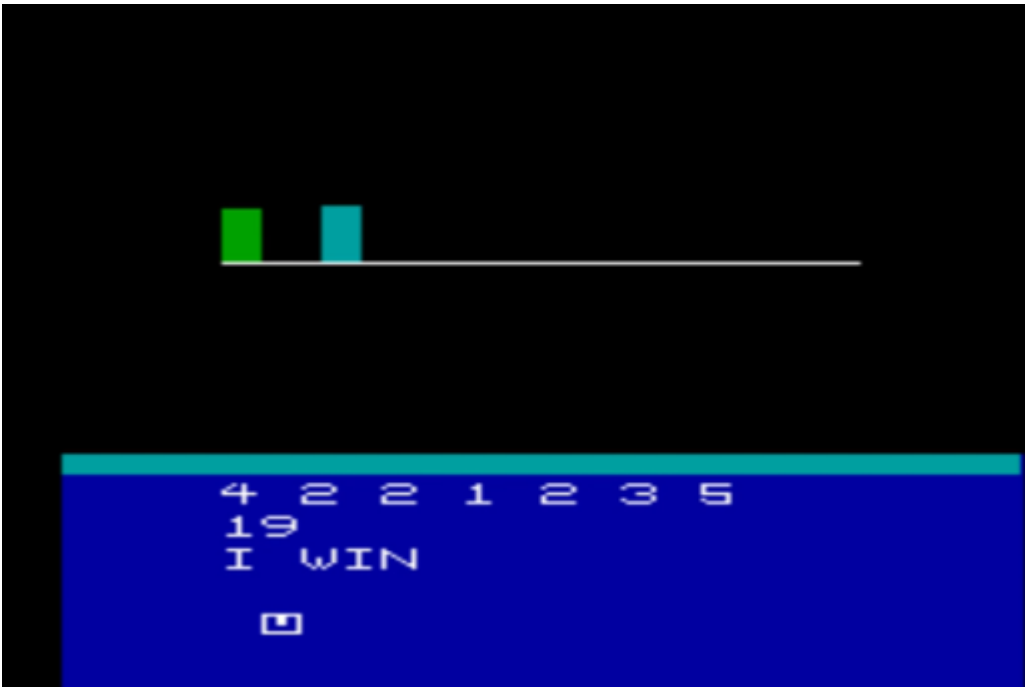
?1
3 3 3 4 5
HIT? 1/0
?0
□



18
4 2
4 2 2



4 4 4
000 000 1
1 00 0





DJ
4 1
HIT? 1/0
?1
G

Another game.





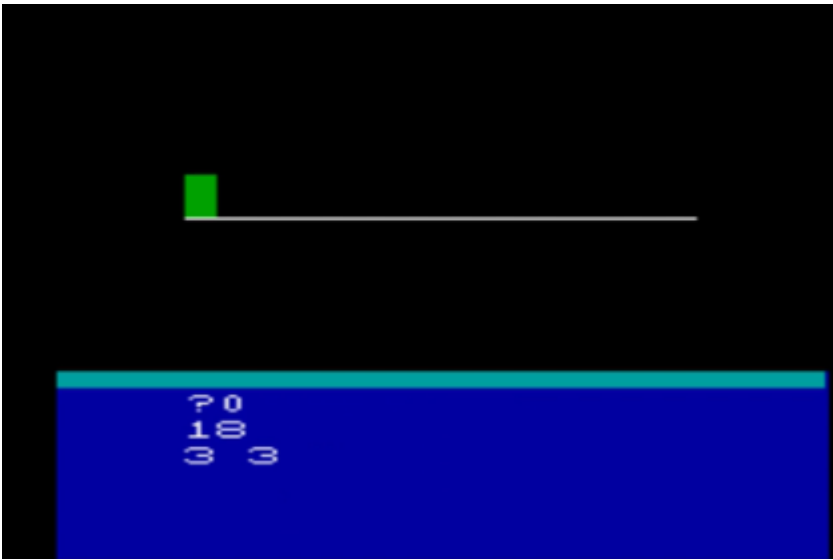
?1
4 1 1 4
HIT? 1/0
?1
□

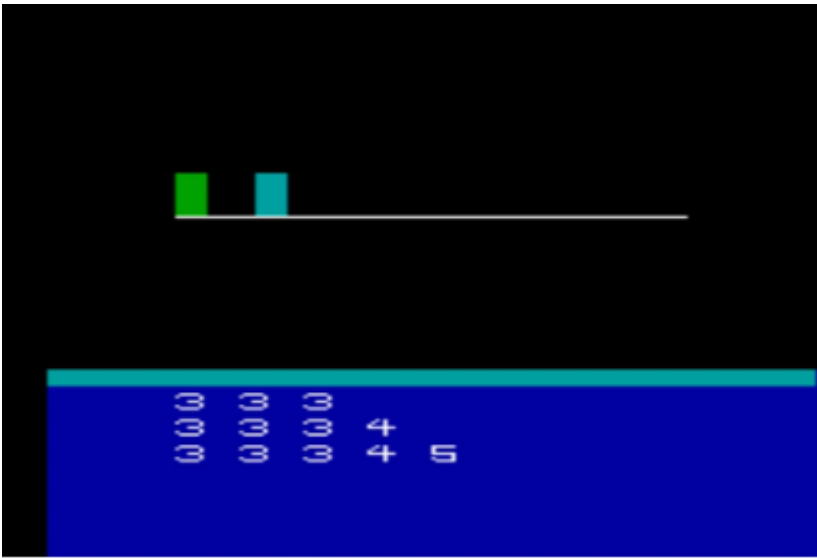


?1
4 1 1 4 3
HIT? 1/0
?1
□



```
?1
4 1 1 4 3 5
HIT? 1/0
?0
  □
```







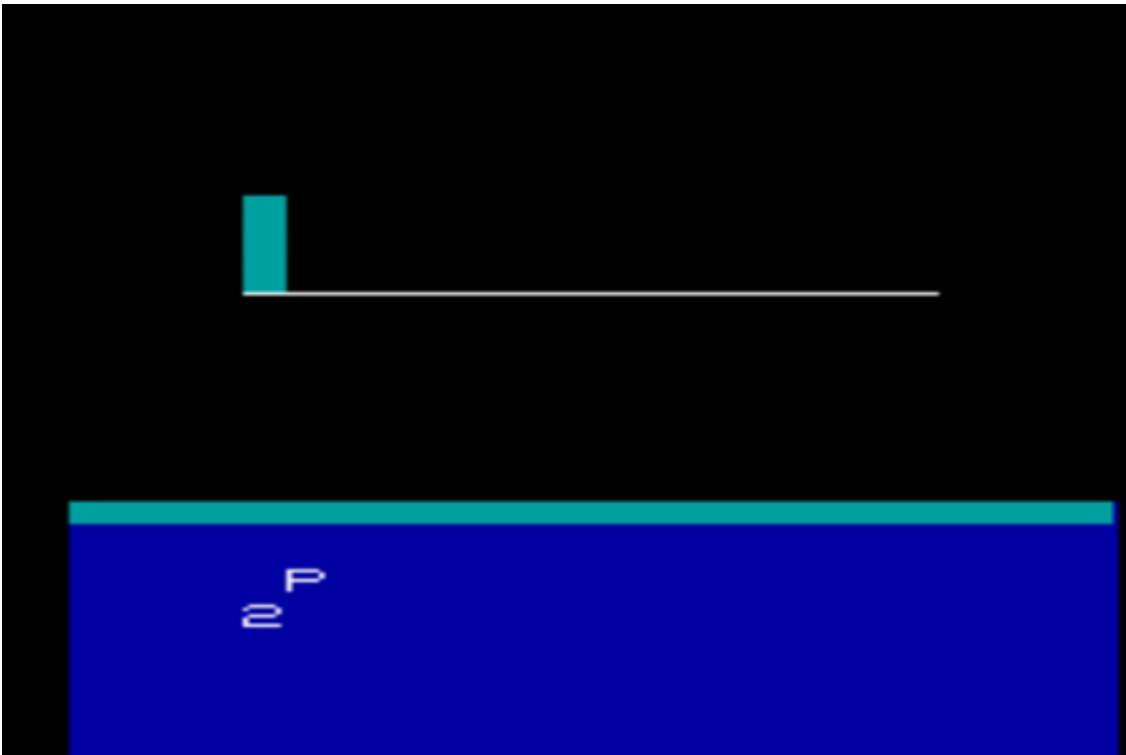


Primes

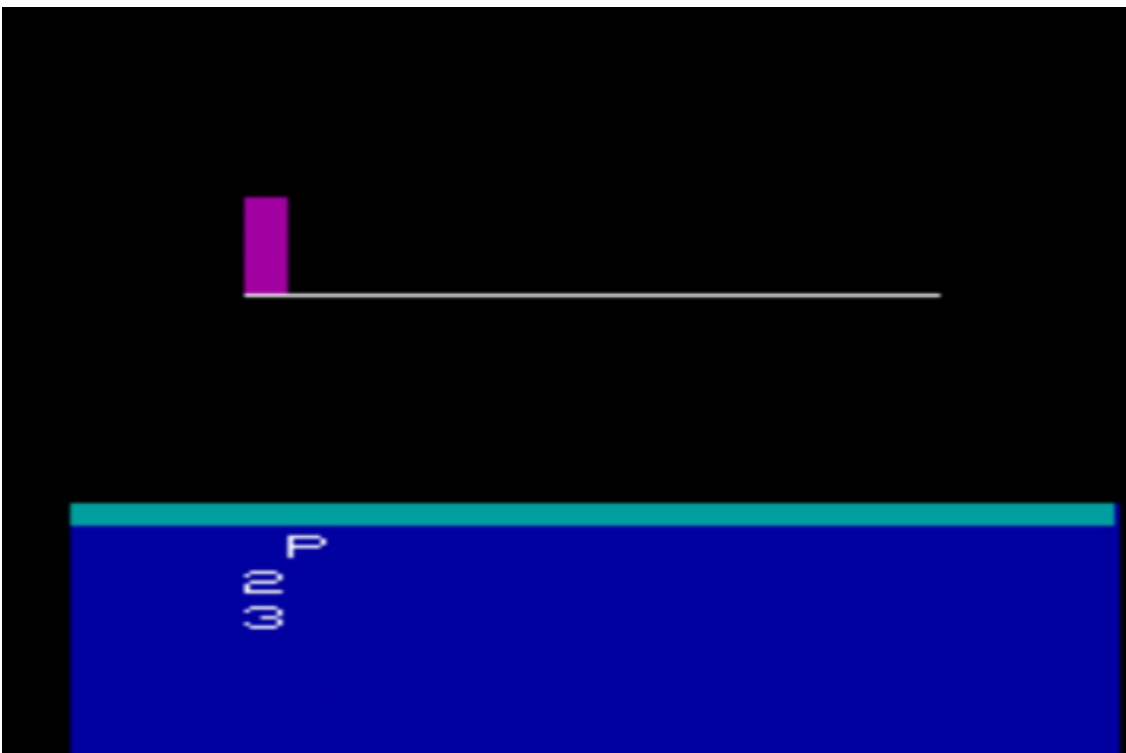
```

PROG P
BARC 2
BARH 30
2
BARC 3
3
I=4
UPTO=23
WHIL I LE UPTO
DO
    PRIM=1
    IF I MOD 2 EQ 0
    THEN PRIM=0
    ELSE
        J=3
        WHIL J LE (I * .5)
        DO
            IF I MOD J EQ 0
            THEN PRIM=0
            EXIT
            ELSE J=J+2
            ENDI
        ENDW
    ENDI
    BARC 1+(I MOD 7)
    IF PRIM EQ 1
    THEN BARH 30
        I
    ELSE
        BARH -30
    ENDI
    I=I+1
ENDW
ENDP
```

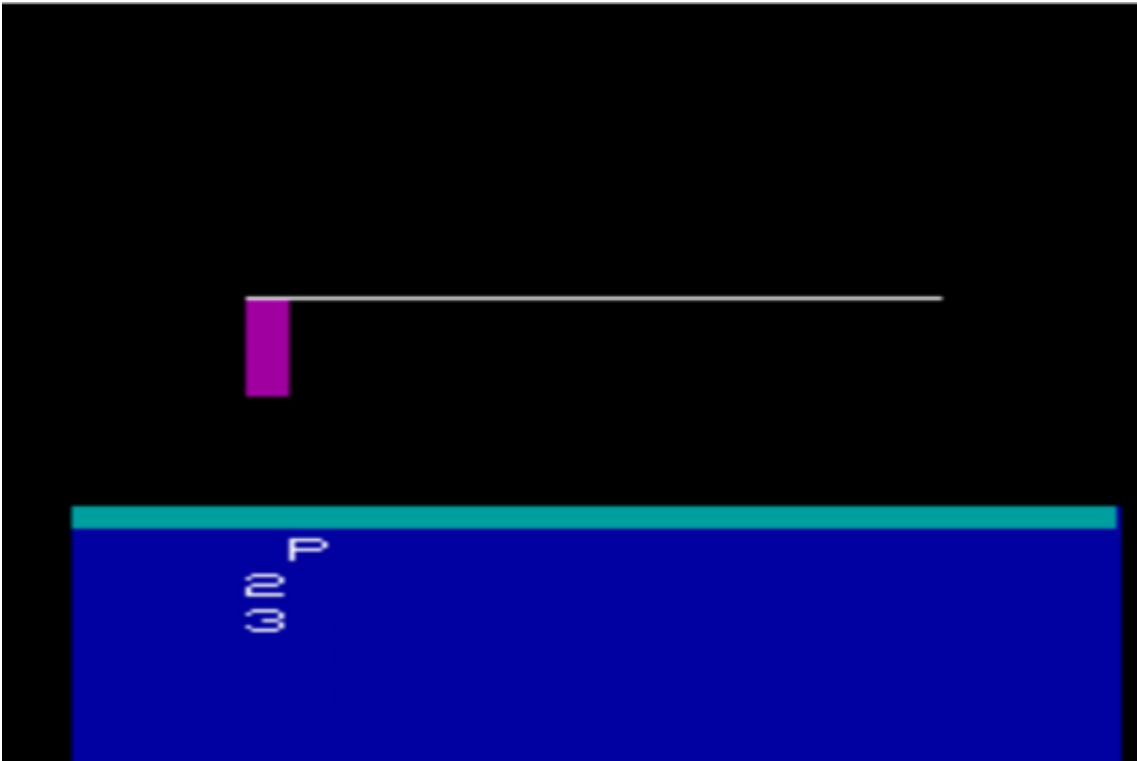
First, consider a program that uses a [primality by trial division algorithm](#).



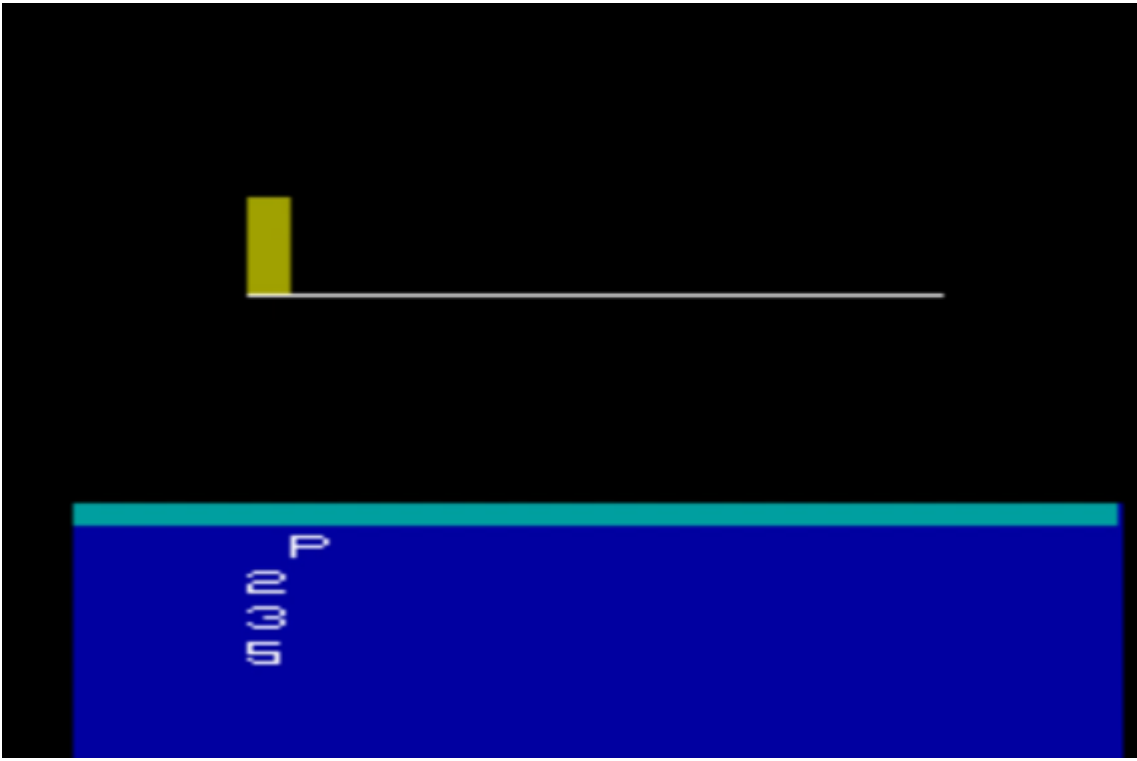
2 is prime, so the flag goes up to indicate a prime was found.



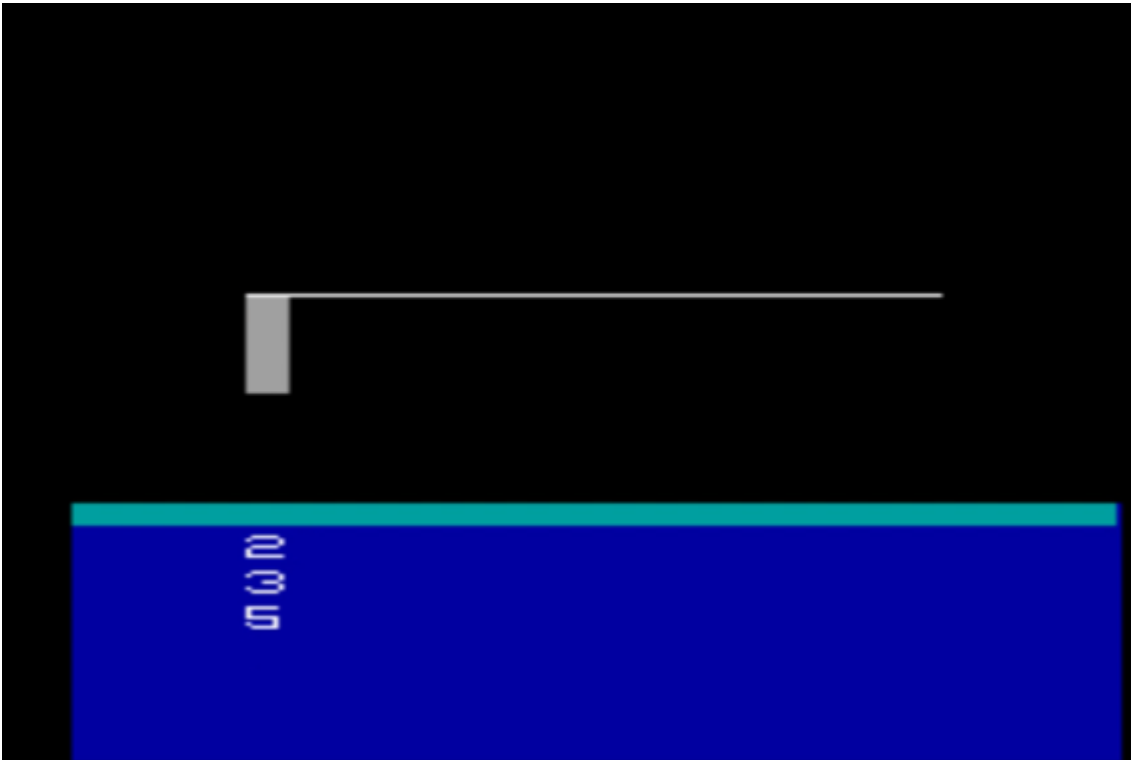
3 is also prime, the flag remains up and shifts color accordingly.



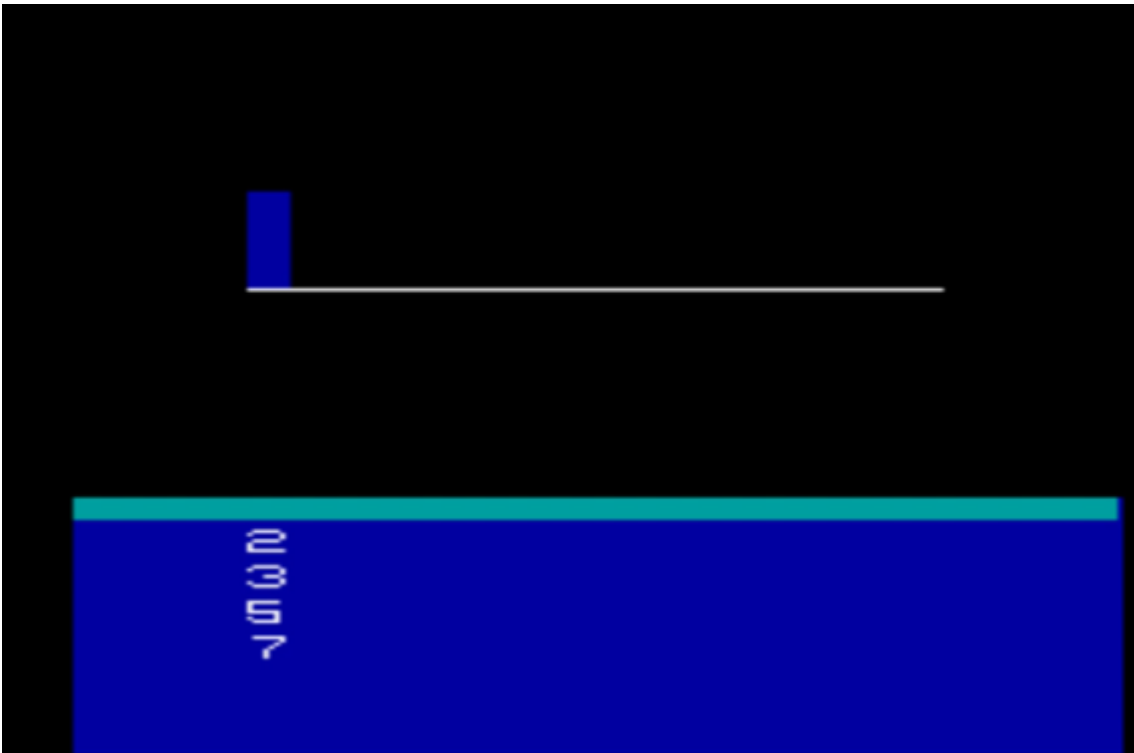
Down for 4.



Up for 5.



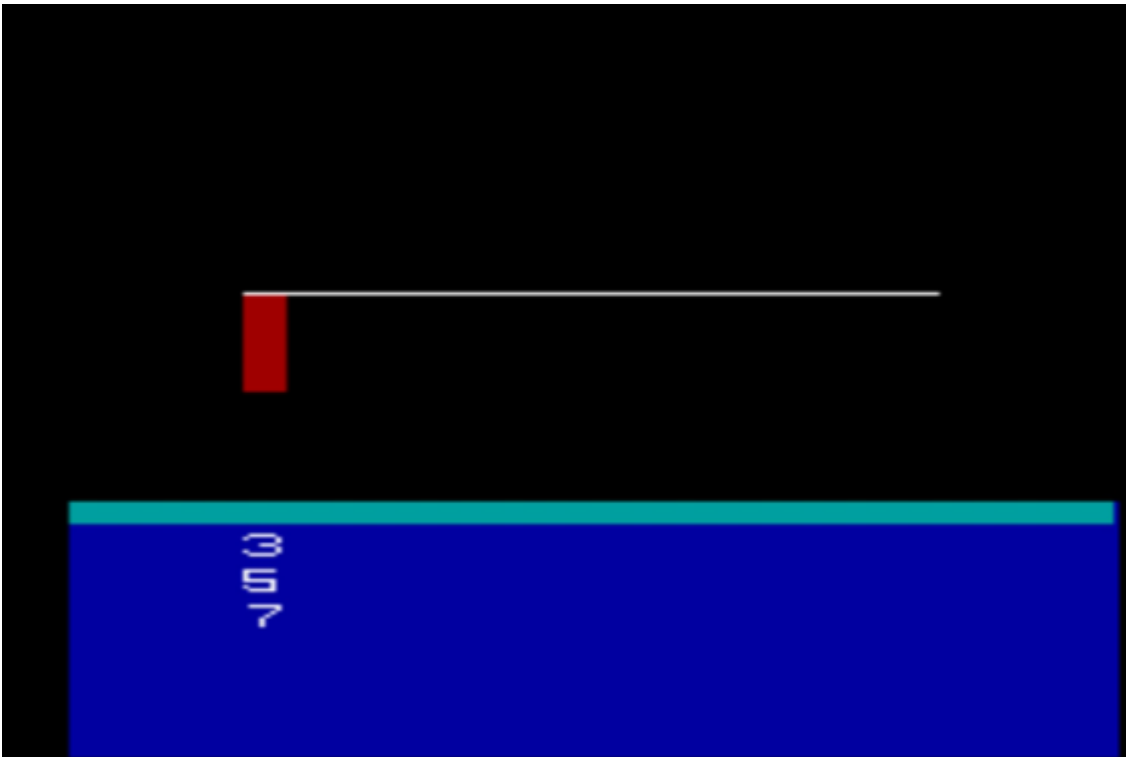
Down for 6.



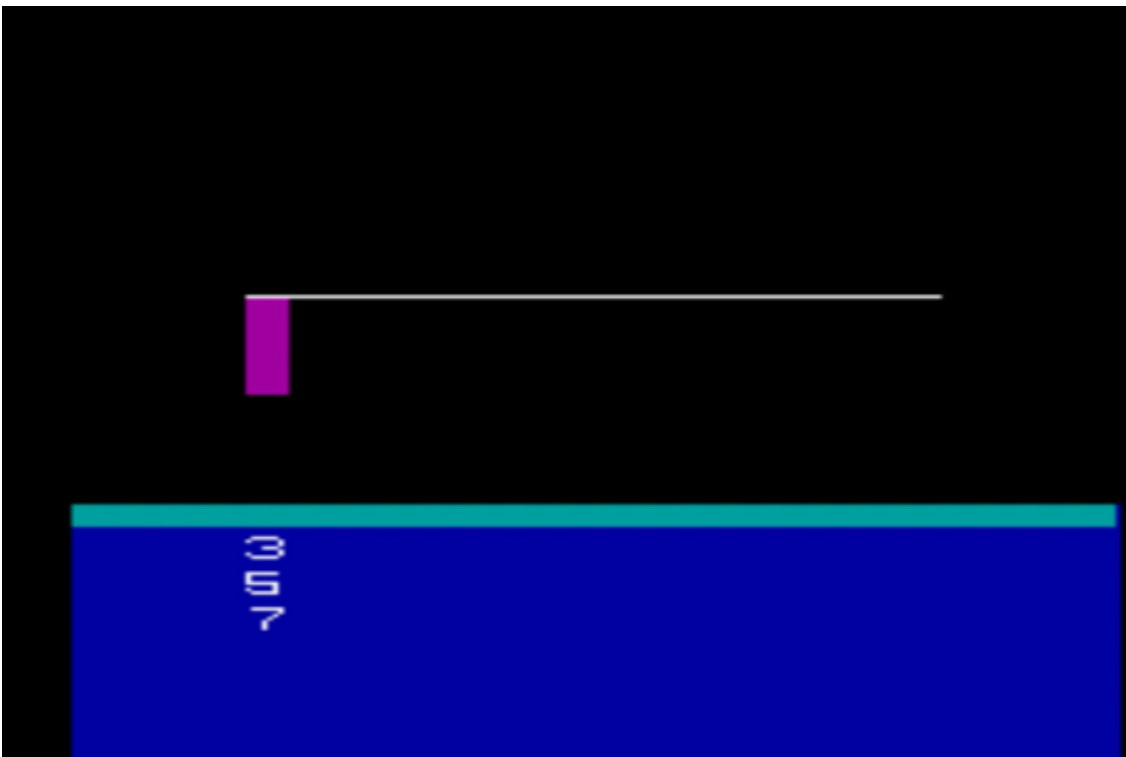
Up for 7.



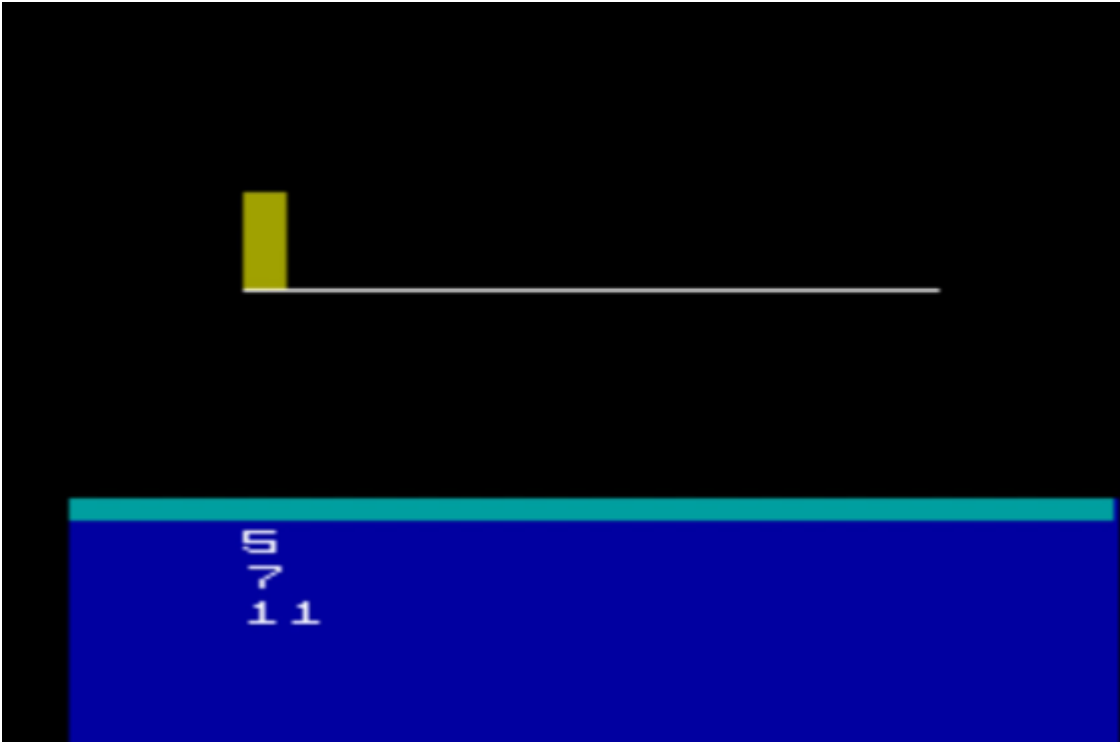
Down for 8.



Down for 9.



Down for 10.



Up for 11. This method is fairly slow. The algorithm requires an increasing amount of time as the numbers being checked grow larger.

Next, consider a different algorithm for finding primes, the [Sieve of Eratosthenes](#).


```
PROG SIEVE
N=INDEX 198
I=2
WHIL I LE SIZE N * .5
DO
    IF N(I) EQ 0
    THEN EXIT
    ELSE
        J=I x I
        WHILE J LE SIZE N
        DO
            N(J)=0
            J=J+I
        ENDW
    ENDI
    I=I+1
ENDW
N
ENDP
```

SIZE N
198
S

To run the sieve for numbers up to and including 198, start by creating an array using index 198.



1	2	3	0	5	0	7	0
0	0	11	0	13	0	0	0
0	17	0	19	0	0	0	0

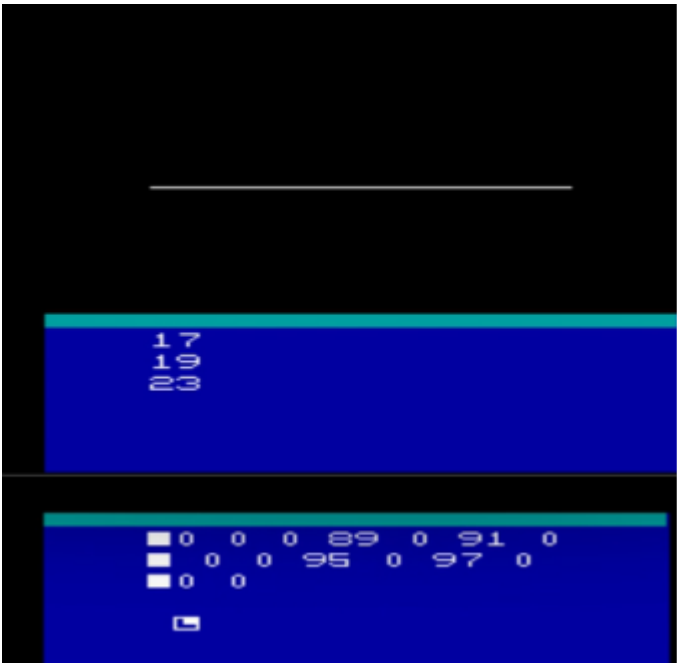
Run the program and the array will be listed, with every number but the primes zeroed out.

■	0	23	0	25	0	0	0
■	29	0	31	0	0	0	
■	35	0	37	0	0	0	4

Only the primes remain when the array is printed.



However, we are constrained by memory, and can only compute primes up to the size of the largest array the system can store.



Comparing the speed of the two algorithms (with the bar chart primality indicators removed), the sieve method can list all primes under 100 before the slower algorithm can list a prime above 23.

37
41
43

■ 0 185 0 187 0
■ 0 0 191 0 193 0
■ 0 0 197 0



It can print all primes up to 198 before the slower method can compute a prime above 43.

Data Statistics

A program to calculate statistics about a data set. For a given set of numbers, the program will list the minimum, maximum, range and sum of the data, the size of the data set, the mean, variance, [standard deviation](#), sample variance, and [sample standard deviation](#).

```
PROG DSTATS
"ENTER YOUR DATA"
X=KEYB
"DATA ENTERED"
X
"MINIMUM"
MIN/X
"MAXIMUM"
MAX/X
"RANGE"
MAX/X - MIN/X
"SUM"
+/X
"SIZE OF DATA SET"
N=SIZE X
N
"MEAN"
M=(+/X)÷N
M
"VARIANCE"
V=+/(X-M)*2)÷N
V
"STD DEV"
V*.5
"SAMPLE VARIANCE"
V=+/(X-M)*2)÷(N-1)
V
"SAMPLE STD DEV"
V*.5
ENDP
```

ENTER YOUR DATA
798 97.5 81.5 76
■ 69

A sample run.

DATA ENTERED
98 97.5 81.5 76
■69

MINIMUM

69

MAXIMUM

98

RANGE

29

SUM

42

SIZE OF DATASET

5

MEAN

84.4

VARIANCE

134.54

STD DEV

11.59914

SAMPLE VARIANCE

168.175

SAMPLE STD DEV

12.96823

□

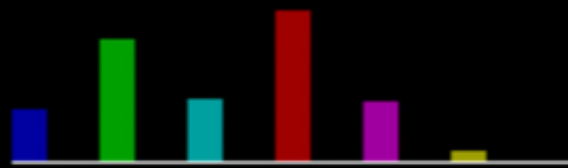
Lotto Picker

Choose five non-repeating numbers between 1 and 69, and an additional bonus number between 1 and 26. In this algorithm we explicitly model each of the 69 main balls in an array and zero them out as they are selected, checking each time if the selected ball is already zeroed or not.

```
PROG LOTTO

// Pick 5 between 1 and 69
// Pick 1 between 1 and 26

B=INDEX 69
P=RAND 26
W=ARRAY 5
C=1
WHILE C LE 5
DO
    R=RAND 69
    IF B(R) NE 0
    THEN
        W(C)=R
        B(R)=0
        C=C+1
    ENDI
ENDW
W,P
BARC INDEX 6
BARH .9x(W,P)
ENDP
```

R
L
22 52 27 65 26 5
□

Sample run.

```
PROG LOTTO2
```

```
// Pick 5 BETWEEN 1 and 69  
// Pick 1 between 1 and 26
```

```
P=RAND 26
```

```
W=ARRAY 5
```

```
C=1
```

```
WHILE C LE 5
```

```
DO
```

```
    R=RAND 69
```

```
    IF (+/ (R EQ W) EQ 0)
```

```
    THEN
```

```
        W(C)=R
```

```
        C=C+1
```

```
    ENDI
```

```
ENDW
```

```
W,P
```

```
BARC INDEX 6
```

```
BARH .9x(W,P)
```

```
ENDP
```

A different method uses add reduction instead, requiring less much storage.

Estimate Pi

This program uses a Monte Carlo algorithm to estimate the value of pi.

```
PROG PI
IN=0
C=1
"HOW MANY ITERATIONS?"
I=KEYB
WHILE C LE I
DO
    X=(RAND 1000000) ÷ 1000000
    Y=(RAND 1000000) ÷ 1000000
    D=(X×X)+(Y×Y)
    IF D LE 1
    THEN IN=IN+1
    ENDI
    C=C+1
ENDW
(4×IN) ÷ I
ENDP
```

```
PI
HOW MANY ITERATI
■ONS?
?100
□
```

```
■ONS?
?100
2.96
□
```

```
PI
HOW MANY ITERATI
■ONS?
?5 0 0
  
```

■ONS?

?500

3.088




```

PROG PI
IN=0
C=1
"HOW MANY ITERATIONS?"
I=KEYB
WHILE C LE I
DO
    X=(RAND 1000000) ÷ 1000000
    Y=(RAND 1000000) ÷ 1000000
    X,Y
    D=(X×X)+(Y×Y)
    IF D LE 1
    THEN IN=IN+1
        BARC C
        BARH 30
    ELSE
        BARC C
        BARH -30
    ENDI
    C=C+1
ENDW
(4×IN) ÷ I
ENDP

```

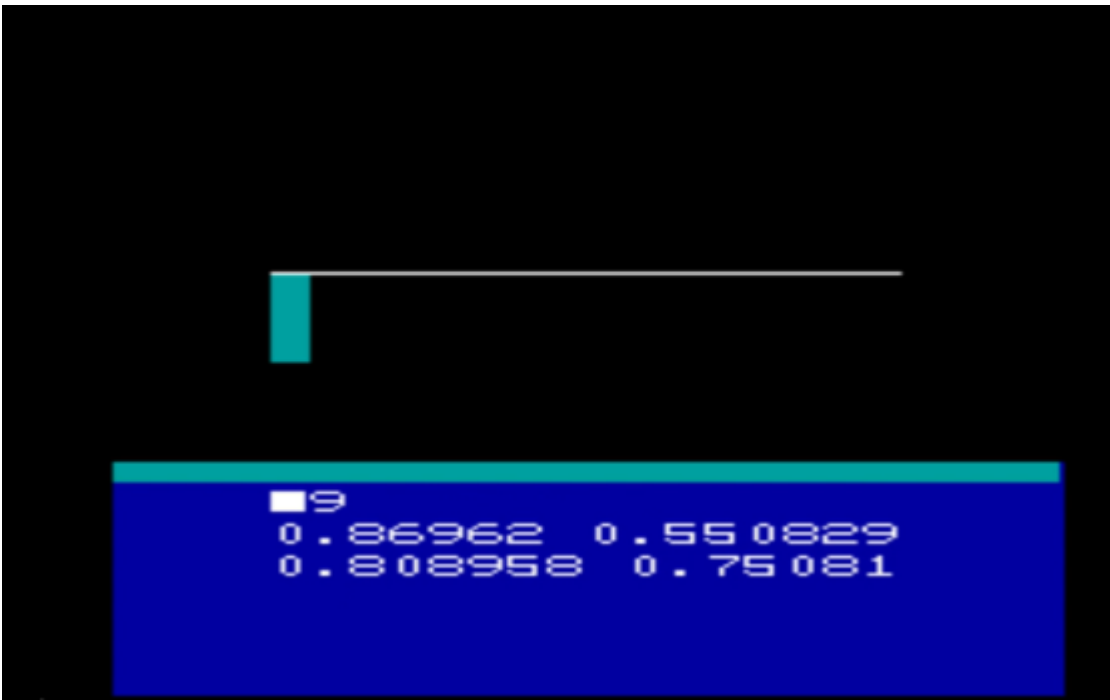
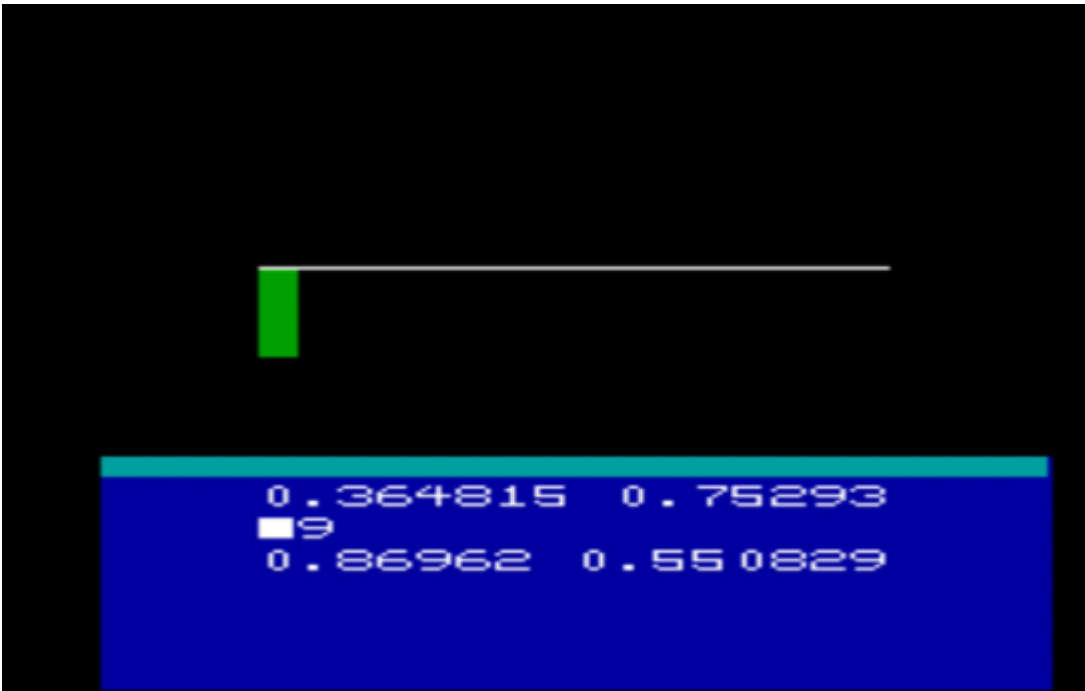
We can use the bar chart to show if the random points fall in the unit circle, to visualize the process.

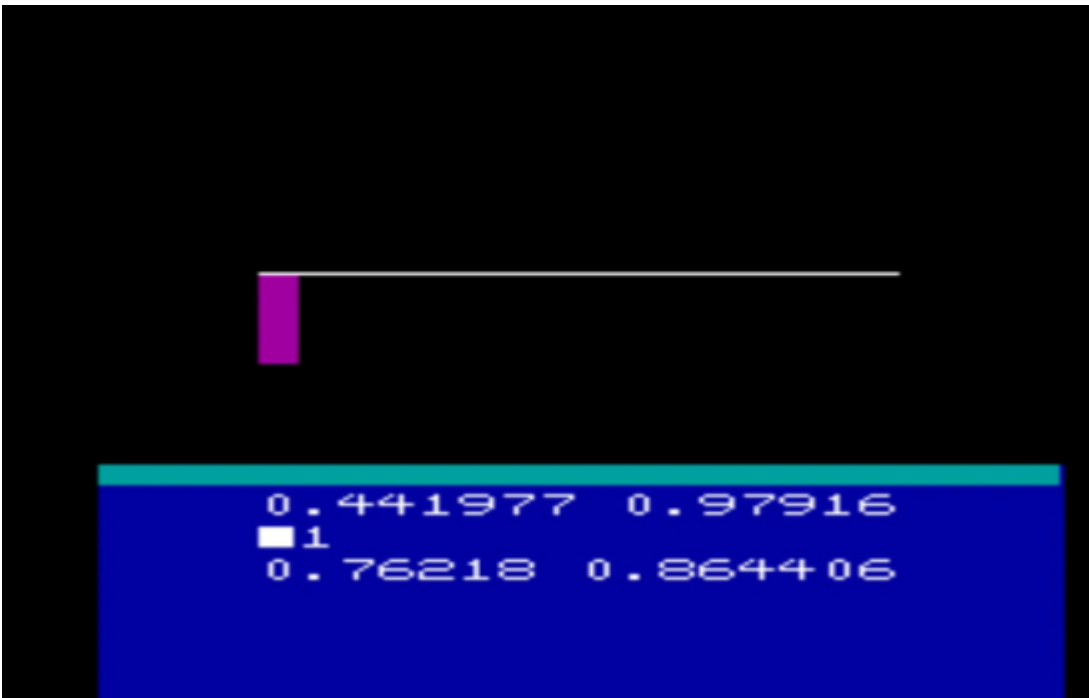
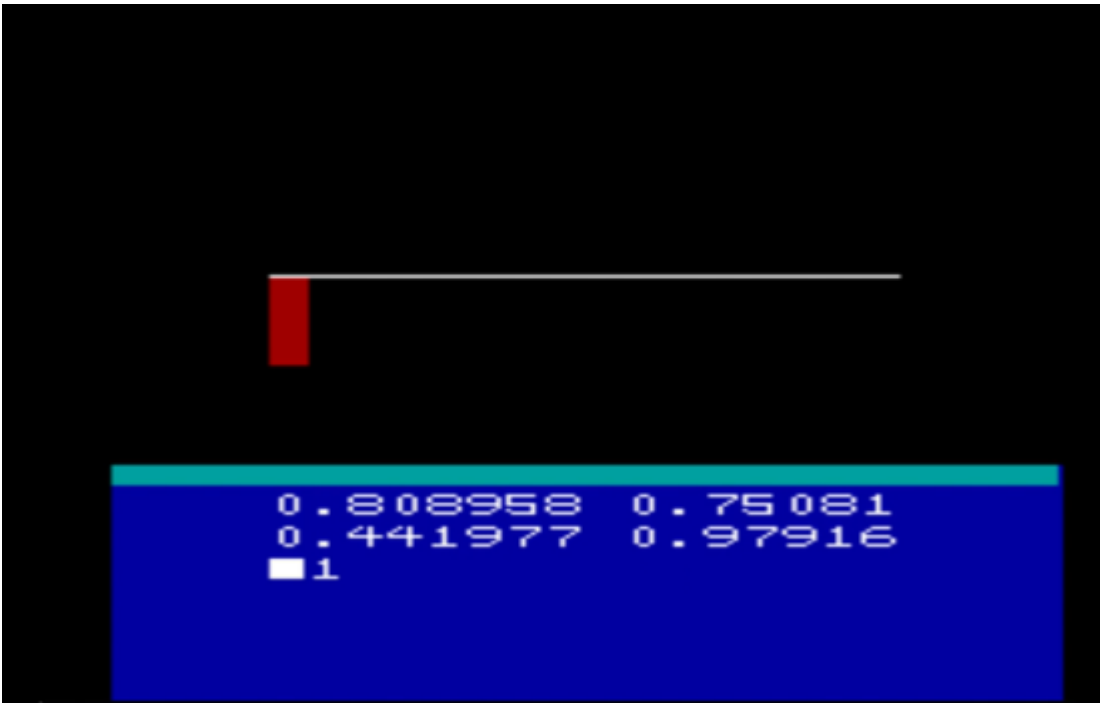
The bar chart value will be positive or negative depending on if the point is in the unit circle.

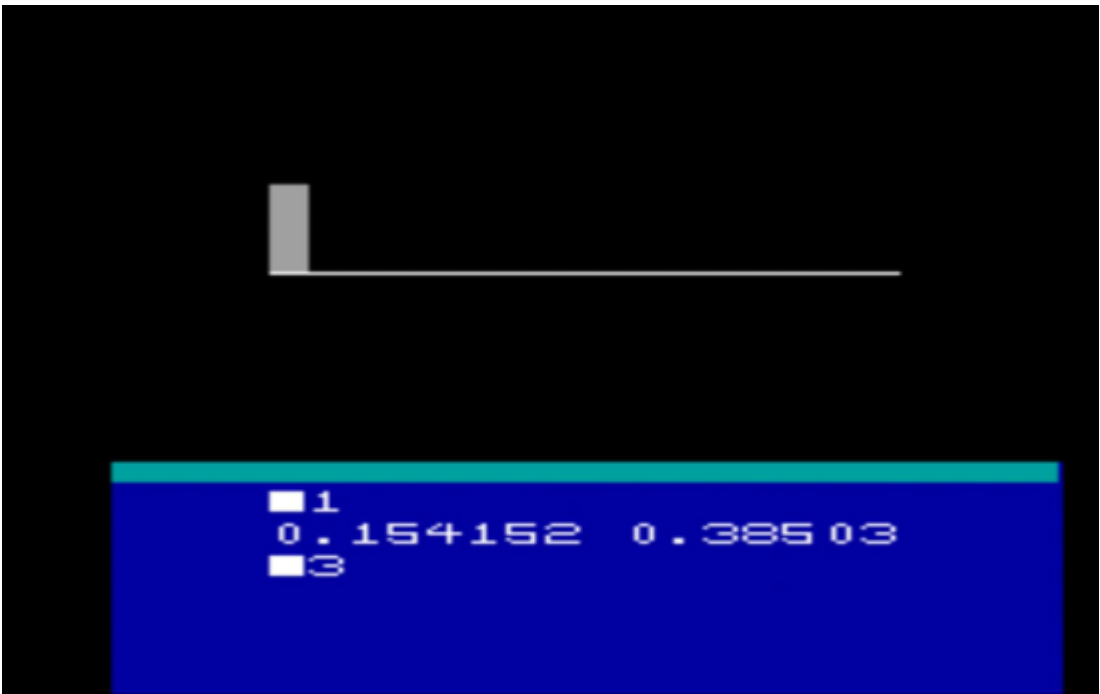
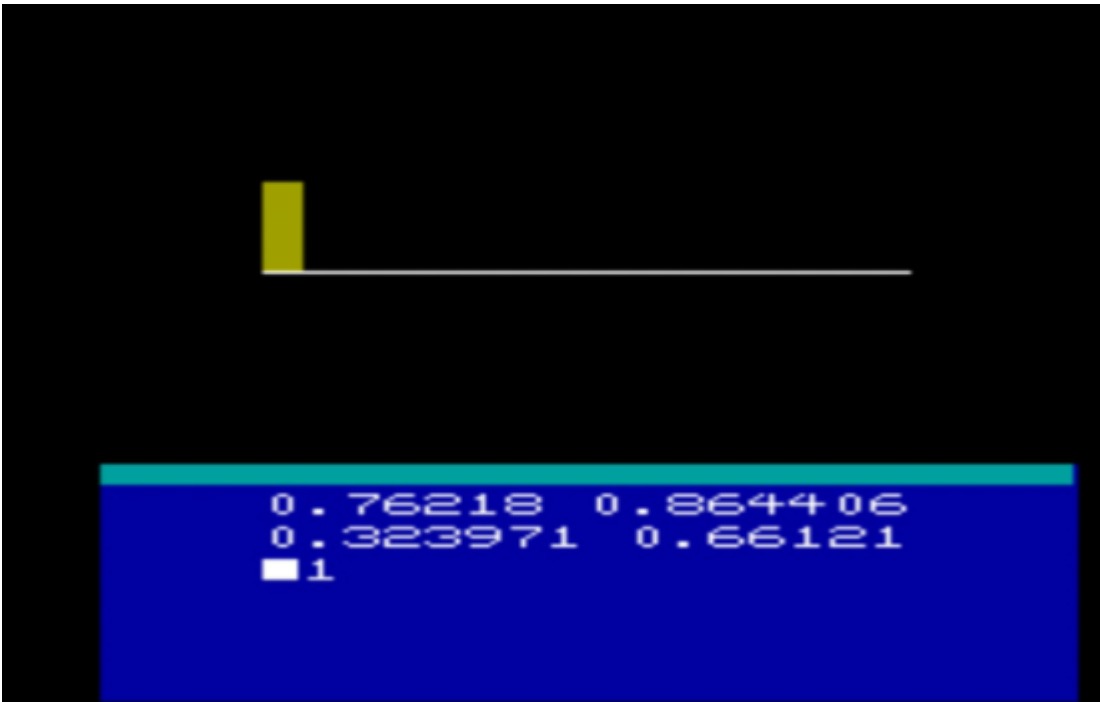
```
PI
HOW MANY ITERATI
■ONS?
?100
  
```



```
?100
0.364815  0.75293
■9
  
```









After 100 points are checked, the estimated value of pi is 3.12.

Taylor, Maclaurin, and Madhava-Leibniz Series

See <https://www.mathsisfun.com/algebra/taylor-series.html>

Exponential

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Sine

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Cosine

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

Arctan (Leibniz Pi)

$$\arctan(1) = \frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$$


```

PROG TAYLOR
X=5
TICK=1
N=22
I=1
E=1
S=0
C=1
A=0
WHIL I LE N
DO
    E=E+(X*I÷!I)
    IF I MOD 2 EQ 0
    THEN
        TICK=-1×TICK
        C=C+(TICK×(X*I÷!I))
    ELSE
        S=S+TICK×(X*I÷!I)
        A=A+TICK×(1÷I)
    ENDI
    I=I+1
ENDW
E,S,C,(A×4)
ENDP

```

T
148.4136 -0.9589
■15 0.2836481 3.
■232315

Running the program for $X=5$, extending out to 22 terms. Arctan is multiplied by 4 to get an approximation of pi.

Stag Hunt

In game theory, the stag hunt models a tension between individual safety and mutual cooperation.

Two hunters independently choose between pursuing a large prey that requires cooperation or settling for a smaller one they can obtain alone. Hunting the larger animal succeeds only if both participants commit. If either defects, the attempt fails. By contrast, the smaller prey can always be caught individually, but yields a much lower payoff. Each participant faces a tradeoff between the certainty of a modest reward and the risk of aiming for a larger shared benefit.

```

PROG STAG
WHIL 1
DO
    BARH 0 0 0
    'STAG OR HARE? 1 OR 0"
    I=KEYB
    IF I EQ 99
    THEN EXIT
    ENDI
    C=-1+RAND 2
    IF I EQ 1
    THEN
        IF C EQ 1
        THEN
            "I CHOSE STAG TOO"
            Q=10
        ELSE
            "I CHOSE HARE."
            Q=1
        ENDI
    ELSE
        IF C EQ 0
        THEN
            "I CHOSE HARE TOO"
            Q=5
        ELSE
            "I CHOSE STAG."
            Q=1
        ENDI
    ENDI
    BARC 2 3 4
    BARH ((-20+I×40),(-20+C×40),(Q×6))
    'YOUR PAYOUT IS"
    Q
ENDW
ENDP

```

```
S
STAG OR HARE? 1
■OR 0
?1
  □
```

A sample run.





STAG OR HARE? 1
■OR 0
?1



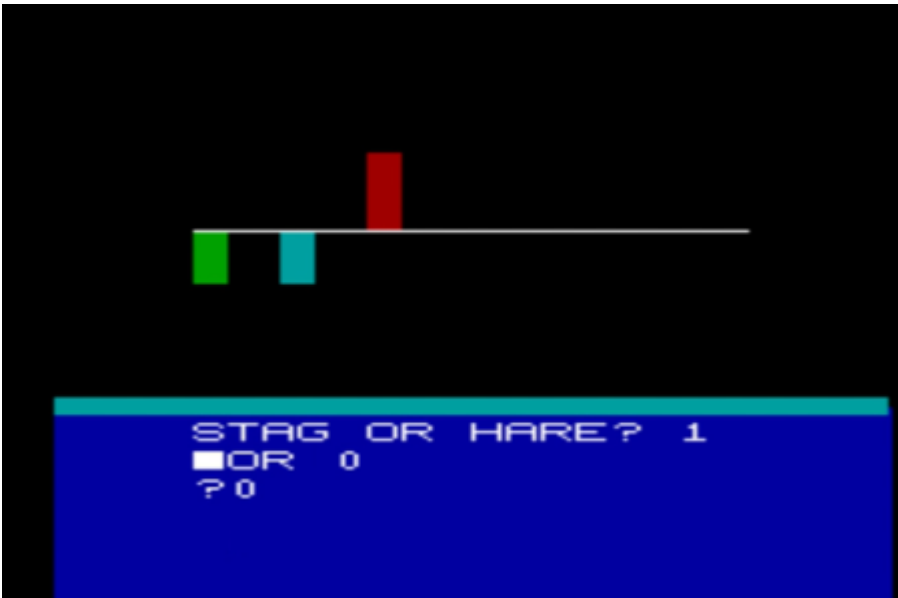
?1
I CHOSE HARE.
YOUR PAYOUT IS
1



STAG OR HARE? 1
■OR 0
? 0
I CHOSE HARE TOO



? 0
I CHOSE HARE TOO
YOUR PAYOUT IS
5





? 0
I CHOSE STAG.
YOUR PAYOUT IS
1

Moo

Moo, or Bulls and Cows, is a classic code-guessing game. The computer chooses a hidden number and the player attempts to discover it through successive guesses, using feedback from each attempt to narrow the possibilities. The game relies entirely on logical deduction rather than chance. As written, the program reveals the code for demonstration purposes, so remove the line where the code is printed to play the game.

The computer selects a three or four digit number with no repeated digits. After the player guesses, the computer reports how many digits are correct and correctly placed (red “bulls”) and how many are correct but in the wrong position (white “cows”).

Long before modern commercial adaptations such as *Mastermind*, the game circulated informally and appeared early in computing culture. During the 1970s it was implemented on time-shared mainframe systems, where it was commonly known as MOO. Variants were ported to Multics and early Unix systems, and versions were distributed through user groups and software libraries for DEC machines. Implementations appeared in several languages, including BASIC and FOCAL, and a PDP-8 version later influenced a commercial version of the game released by Milton Bradley.

```

PROG MOO
WHIL 1
DO
  Z= RAND 5 5 5
  IF +/(Z(1) EQ Z) EQ 1
  THEN
    IF +/(Z(2) EQ Z) EQ 1
    THEN
      EXIT
    ENDI
  ENDI
ENDW
"GUESS THE 3 NUMBERS"
WHIL 1
DO
  I=KEYB
  IF I(1) EQ 99
  THEN
    EXIT
  ENDI
  X=+/(Z EQ I)
  IF 3 EQ X
  THEN "YOU WIN"
  EXIT
  ELSE
    C=X
    J=1
    R=0
    WHIL J LE 3 //CORRECT COLOR WRONG SPOT
    DO
      IF Z(J) NE I(J)
      THEN
        R=R+ (+/(Z(J) EQ I ) )
      ENDI
      J=J+1
    ENDW
    "WHITE/RED"
    C,R
  ENDI
ENDW
ENDP

```



P1 4 3
WHITE/RED
1 0
P2 5 1

U

P4 5 1
WHITE/RED
0 2
P5 6 4

U



R5 2 4
WHITE/RED
1 2
R5 4 2

U



```

PROG MOO4
WHIL 1
DO
  Z= RAND 5 5 5 5
  IF +/(Z(1) EQ Z) EQ 1
  THEN
    IF +/(Z(2) EQ Z) EQ 1
    THEN
      IF +/(Z(3) EQ Z) EQ 1
      THEN
        EXIT
      ENDI
    ENDI
  ENDI
ENDI
ENDW
"GUESS THE 4 NUMBERS"
WHIL 1
DO
  I=KEYB
  IF I(1) EQ 99
  THEN
    EXIT
  ENDI
  X=+/(Z EQ I)
  IF 4 EQ X
  THEN "YOU WIN"
  EXIT
  ELSE
    C=X
    J=1
    R=0
    WHIL J LE 4 //CORRECT COLOR WRONG SPOT
    DO
      IF Z(J) NE I(J)
      THEN
        R=R+(+/(Z(J) EQ I ))
      ENDI
      J=J+1
    ENDW
    "WHITE/RED"
    C,R
  ENDI
ENDW
ENDP

```



P4 9 7 8

WHITE/RED

0 0

P1 2 3 4

u





R5 R1 R2
W1 R1 R2
R3
P

4

PS 2 1 3

WHITE/RED

2 2

P

4



Ecte's Guess

A port of David Ahl's [GUESS](#). Dedicated to my friend Ecte, who enjoys porting this game to new languages.

```

PROG GUES
BARC 3
  "WHAT LIMIT DO YOU WANT"
  LIM=KEYB
  COUNT=1
  NUM=RAND LIM
  "OK I CHOSE A NUMBER. WHAT IS YOUR GUESS"
  WHILE NUM NE G=KEYB
    DO COUNT=COUNT+1
      IF G LT NUM
        THEN "TOO LOW"
          BARH -30
        ELSE "TOO HIGH"
          BARH 30
        ENDIF
      "GUESS AGAIN"
    ENDWHILE
  BARH 0
  "YOUR SCORE VERSUS BINARY SEARCH"
  COUNT, (1+ FLOOR (LOG LIM ÷ LOG 2))
ENDP

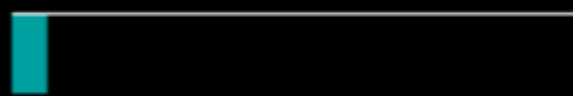
```

GUES
WHAT LIMIT DO YO
■U WANT?
?100
□

OK I WILL CHOSE
■ A NUMBER. WHAT
■ IS YOUR GUESS
?
□



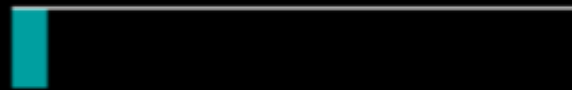
?26
TOO HIGH
GUESS AGAIN
?12
□



?12
TOO LOW
GUESS AGAIN
?19
□



?19
TOO HIGH
GUESS AGAIN
?15
□



?15
TOO LOW
GUESS AGAIN
?17
□



?17
TOO HIGH
GUESS AGAIN
?16
□

?16
YOUR SCORE VERSU
■S BINARY SEARCH
6 7

Recursive Coin Change Maker

This program counts out coins to give as change.

Outside the program, set $Q=D=N=P=0$, and set X to the amount of money to count.

Q=D=N=P=0

X=89

PROG C

IF X GE 25

THEN Q=Q+1

 X=X-25

ELSE

 IF X GE 10

 THEN D=D+1

 X=X-10

 ELSE

 IF X GE 5

 THEN N=N+1

 X=X-5

 ELSE

 P=X

 X=0

 ENDI

 ENDI

ENDI

"X Q D N P"

X,Q,D,N,P

IF X GT 0

 THEN C

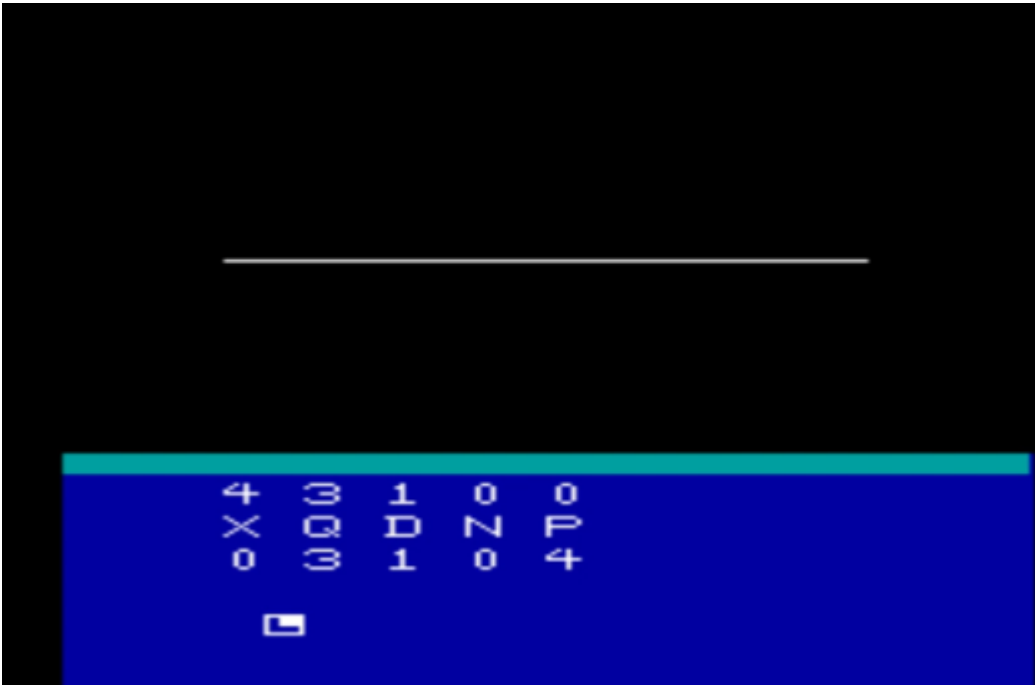
ENDI

ENDP

X=89
Q=D=N=P=0
C
□

C	X	Q	D	N	P
64		1	0	0	0

64	1	0	0	0
X	Q	D	N	P
39	2	0	0	0



X=41
CQ=D=N=P=0
C
□

C	X	Q	D	N	P
16	1	0	0	0	

S	1	1	0	0
X	Q	D	Z	P
1	1	1	1	0

1	1	1	1	0
X	Q	D	Z	P
0	1	1	1	1



Calendar Clock

Enter in the program, but before running it, first assign values for the year, month, day, hour, minute, and second variables. Set the bar chart colors, too.

```
BARC INDEX 7
```

```
Y=25
```

```
O=12
```

```
D=31
```

```
H=23
```

```
M=59
```

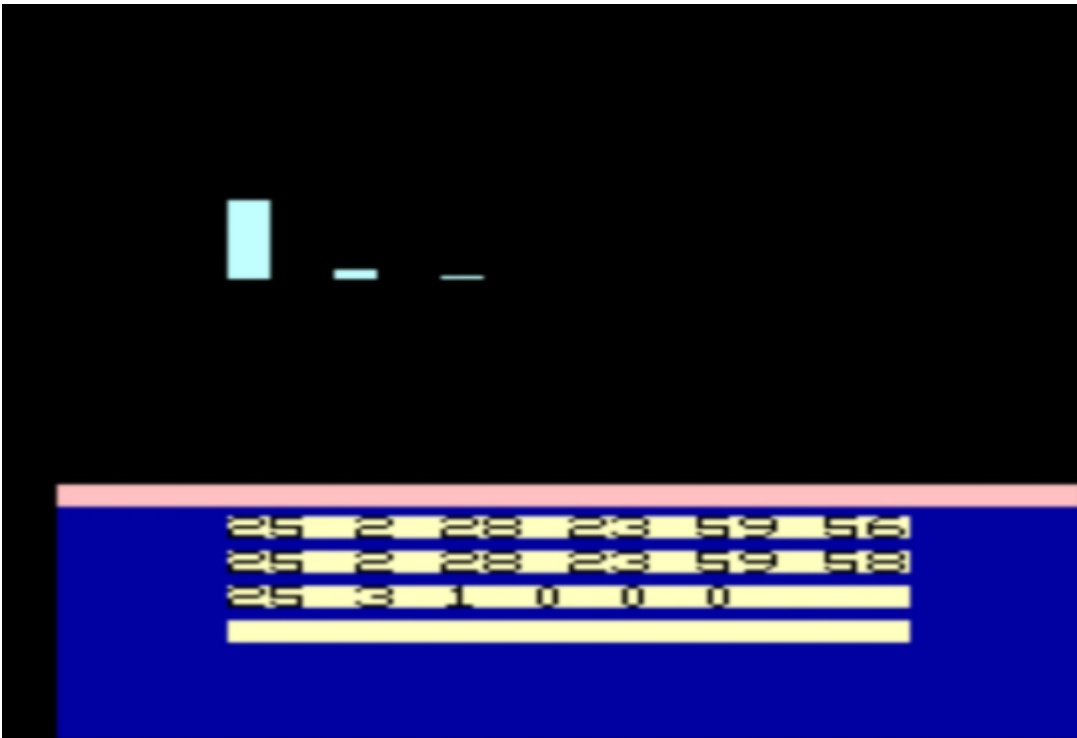
```
S=50
```

For example, these variables are for New Years Eve 2025, a ten second countdown to 2026.

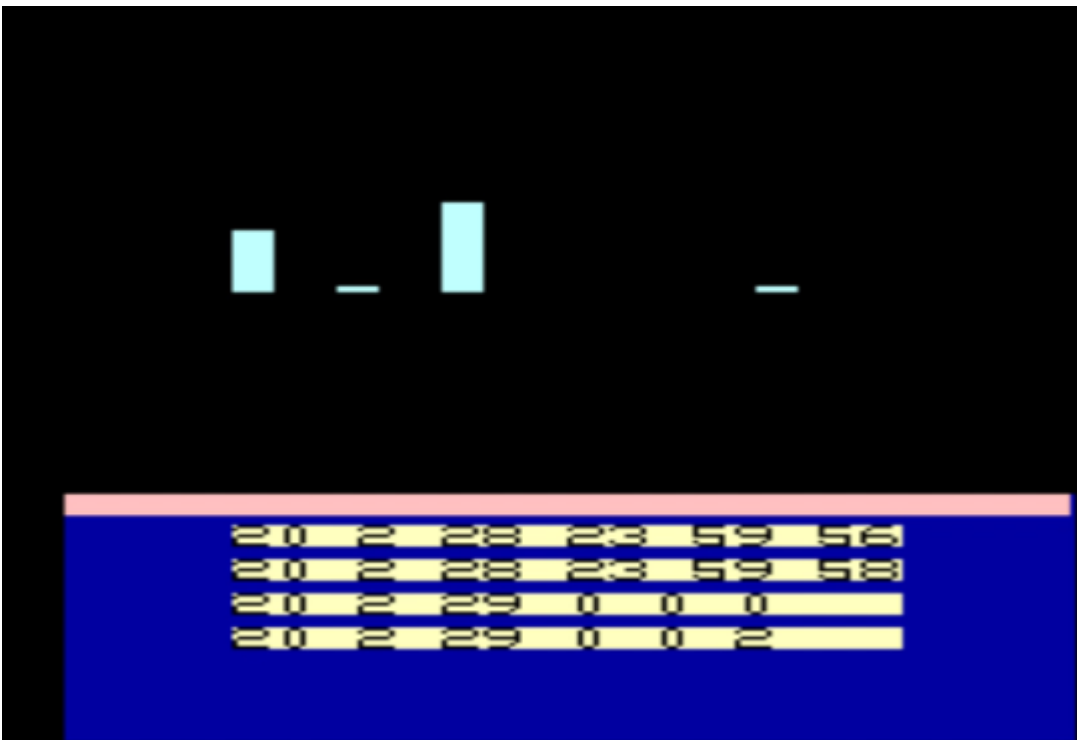
```

PROG CLOCK
WHIL 1
DO
    S=S+2
    IF S GE 60
    THEN S=0
        M=M+1
    ENDI
    IF M EQ 60
    THEN M=0
        H=H+1
    ENDI
    IF H EQ 24
    THEN H=0
        D=D+1
    ENDI
    IF (D EQ 30) AND (O EQ 2)
    THEN D=1
        O=3
    ENDI
    IF (D EQ 29) AND (O EQ 2) AND ((Y MOD 4) NE 0)
    THEN D=1
        O=3
    ENDI
    IF (D EQ 31) AND (+/(O EQ 4 6 8 9 11))
    THEN D=1
        O=O+1
    ENDI
    IF D EQ 32
    THEN D=1
        O=O+1
    ENDI
    IF O EQ 13
    THEN O=1
        Y=Y+1
    ENDI
    BARH Y,O,D,H,M,S
    Y,O,D,H,M,S
ENDW
ENDP

```



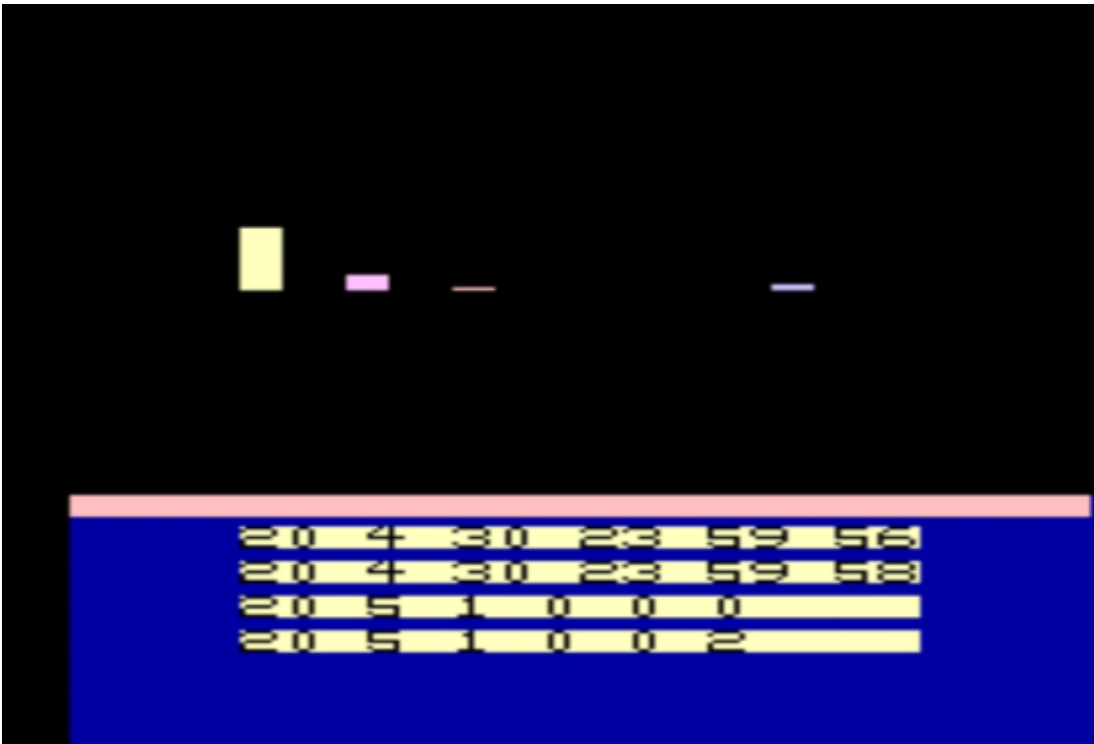
A non-leap year.



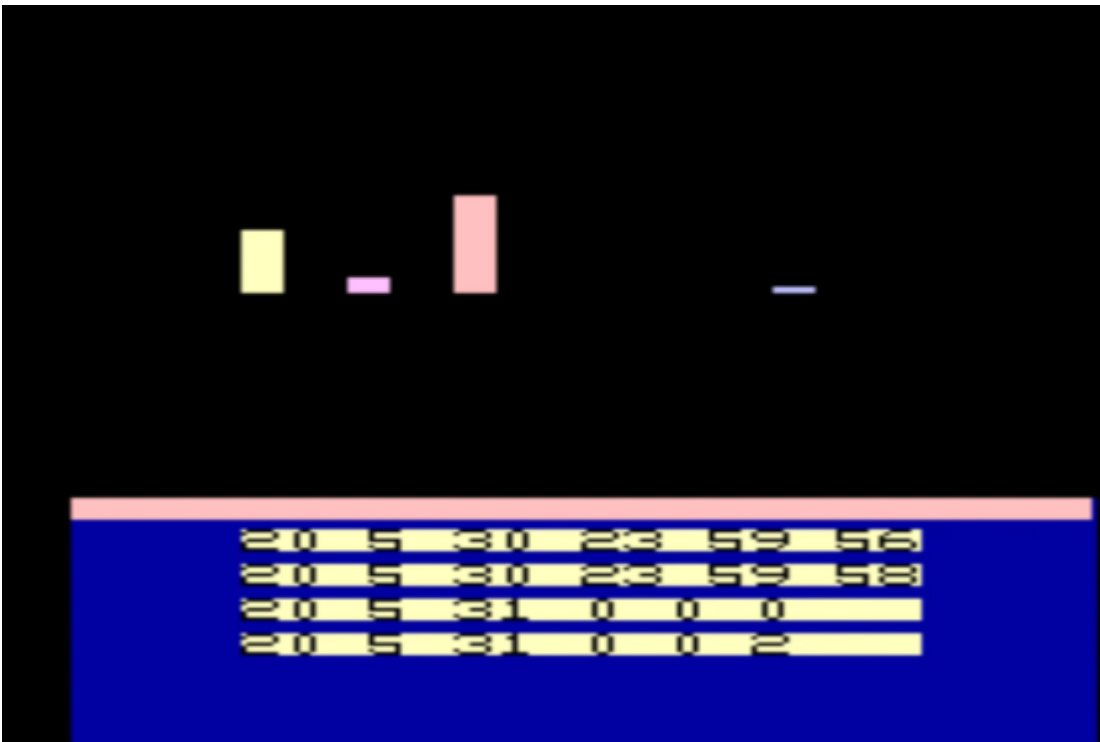
A leap year.



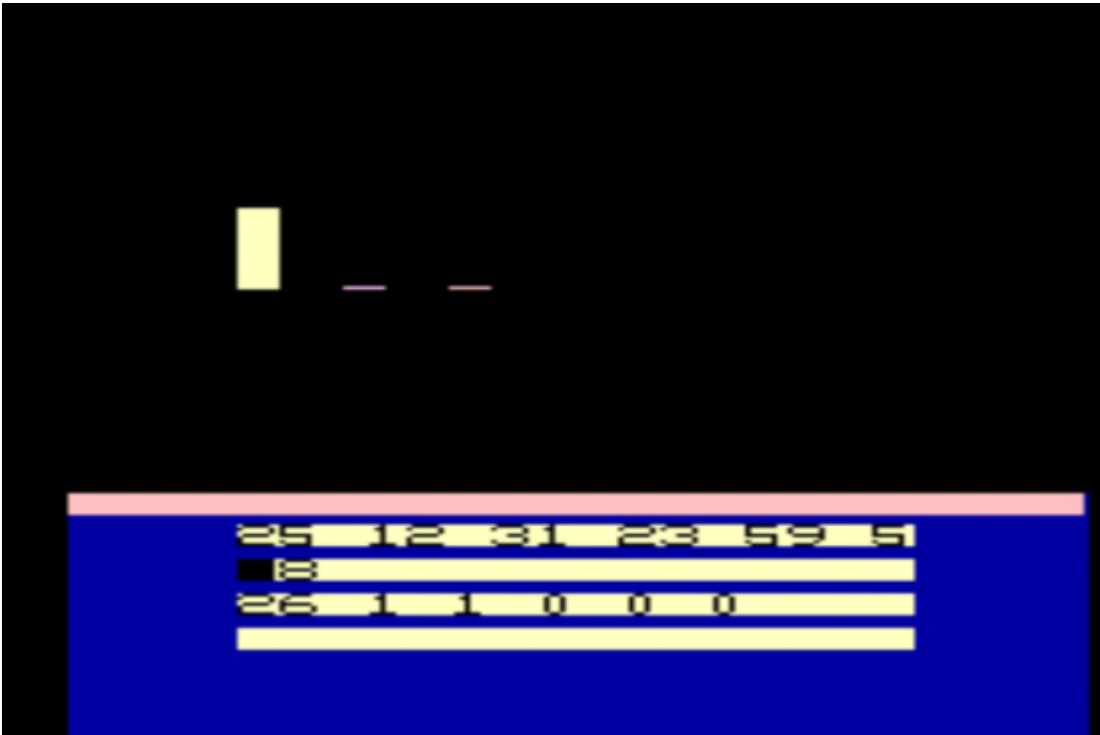
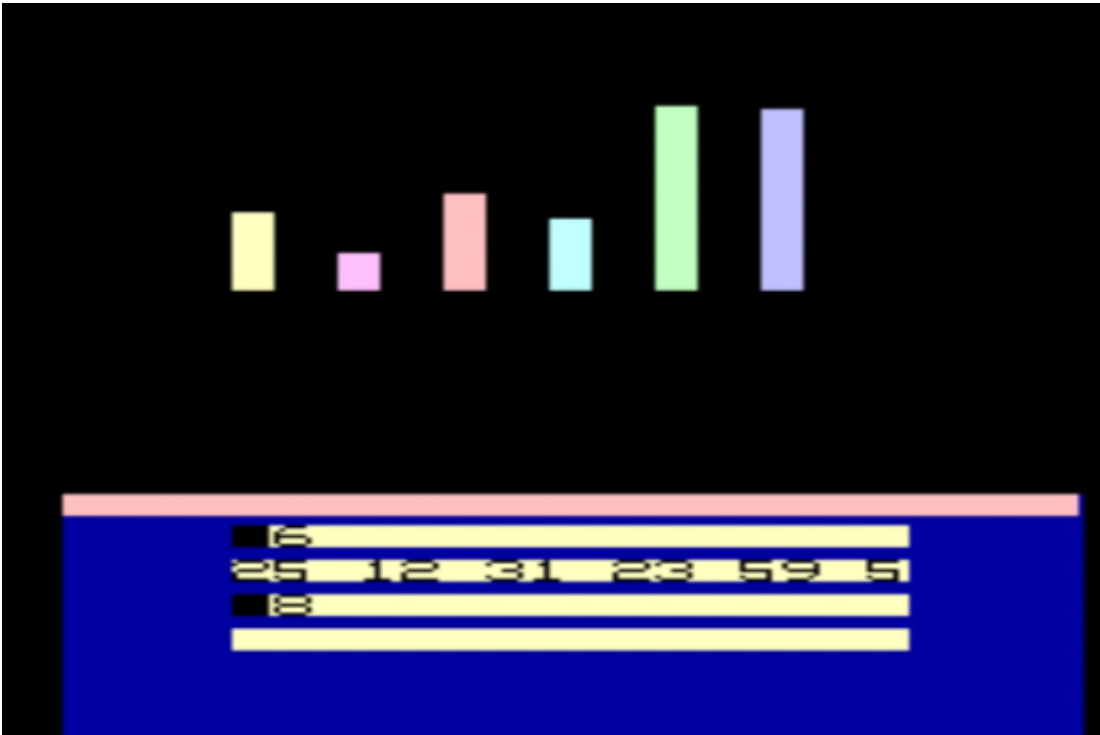
20	2	29	23	59	56
20	2	29	23	59	58
20	3	1	0	0	0
20	3	1	0	0	2

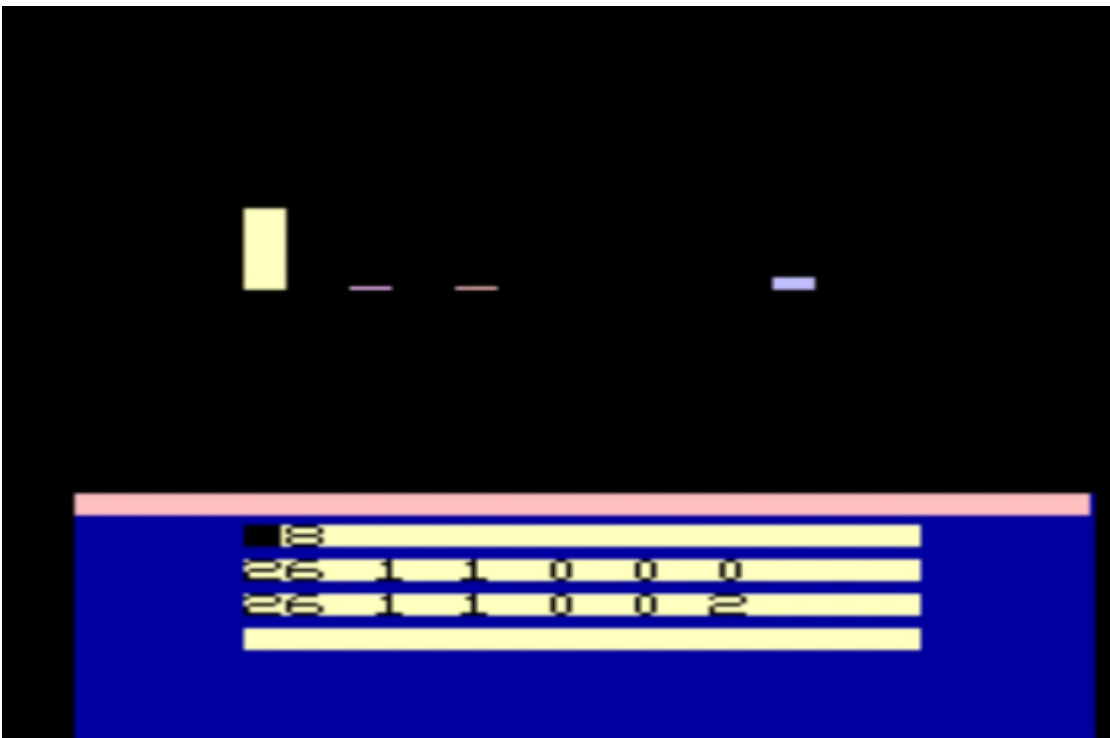


April has 30 days.



May has 31.





Happy New Year!

Credits

All material from works authored or edited by David Ahl is in the public domain.

See <https://blog.adafruit.com/2022/06/16/david-ahl-places-all-his-classic-computing-publications-into-the-public-domain/>



The APL/S games in this book are collected here:

<https://github.com/jamescprice/21-APL-Games-and-One-Liners-For-Your-Umtech-VideoBrain/blob/main/21games.zip> – All the games in this book.

Preservation copies of the VideoBrain APL/S cartridge ROM are maintained by third-party retrocomputing archives for historical reference and research use. This publication includes APL/S program listings in printed form but does not include ROM images. For discussion see:

<https://groups.io/g/Channel-F-and-VideoBrain/topic/101298602>

Documentation of The Computational Language APL/S:

<https://aplwiki.com/wiki/APL/S> Great technical description of the language at APL Wiki

<https://archive.org/details/byte-magazine-1979-12/page/n81/mode/2up>
“APL/S: An Alternative” by Robert G. Brown BYTE magazine 12/1979

<https://archive.computerhistory.org/resources/access/text/2024/06/102804524-05-001-acc.pdf> – Two papers: "An Introduction to APL/S". Conference Proceedings of the Third West Coast Computer Faire” by Robert G. Brown p247 and “VideoBrain and the APL/S Language: First Pass at Everyman’s Computer” by Ted Haynes p246

<https://archive.org/details/apls-brochure/APLS-Brochure/> – Includes Lunar Lander game <https://forum.vcfed.org/index.php?threads/i-have-been-blessed-with-videobrain-documentation.80042/> - Links to zip file with great documentation and schematics

<https://forum.vcfed.org/index.php?threads/videobrain.71462/page-2> – Lots of documentation here!

Audio Discussion:

https://archive.org/details/camvchm_000053/camvchm_000053_t01_a_access.mp3 - "This is the recording of the session titled 'High level languages for micros,' and the following papers were presented: 'VideoBrain and the APL/S language: First pass at everyman's computer,' presented by Ted Haynes. 'An introduction to APL/S: A modern computational language for personal computing,' presented by Robert G. Brown."

<https://exhibits.stanford.edu/silicongenesis/catalog/nc497jv0418> – An interview with Albert Yu. 00:12:06 Discusses leaving Intel for Video Brain, where he developed microprocessors for use in consumer electronics. 00:17:52 Discusses leaving Video Brain in 1979 to sell home computers between the United States and China.

<https://randoc.wordpress.com/category/umtech-videobrain/> - Links to audio of conference talk

<https://www.orphanedgames.com/videobrain/>

More links:

http://hcvgm.org/Static/Manuals/VideoBrain/VB_Unit_Manual.pdf - System Manual with image of keyboard keys -

<https://randoc.wordpress.com/category/umtech-videobrain/> - A good discussion with many links

<https://www.atariprotos.com/othersystems/videobrain/cartridges/apls/apls.htm>

"The existence of APL/S has been speculated for years. More than one person has claimed to own the cartridge back in the day and it was mentioned in several advertisements, but no actual APL/S cartridge has ever been found. However in 2021 the first evidence of APL/S's existence turned up in the form of a manual in an eBay auction. Unfortunately the cartridge and box were nowhere to be seen. This all changed in 2023 when a complete copy of APL/S was finally located. Only 91 APL/S cartridges were ever made making it the rarest non-prototype VB cartridge."

<http://hcvgm.org/VideoBrain.html> - A keyboard diagram, machine specs

http://blog.kevtris.org/blogfiles/videobrain/videobrain_unwrapped.txt – A hardware deep-dive

<https://www.oldcomputers.net/videobrain.html> – Great images of system internals

https://www.ballyalley.com/articles_and_news/creative_computing_article.pdf – Two Creative Computing articles

Don't miss out!

Click the button below and you can sign up to receive emails whenever James Price publishes a new book. There's no charge and no obligation.

Sign Me Up!

<https://books2read.com/r/B-A-GJGCE-SIVNI>

BOOKS  READ

Connecting independent readers to independent writers.